CittàStudiEdizioni

MARZIA BONECCHI e LUISA ZAULI
PROGRAMMARE IN QUICKBASIC

Il testo nasce dall'esigenza di fornire agli studenti del corso di laurea in Matematica dell'Università degli Studi di Milano, al loro primo impatto con gli elaboratori, un buon supporto didattico atto a renderli il più possibile autonomi nell'uso degli strumenti a loro disposizione. Esso può comunque essere utilizzato con profitto anche da tutti coloro, studenti e no, che vogliono impradronirsi velocemente del linguaggio Basic e dell'ambiente QuickBASIC.
Dopo una panoramica dei principali comandi del sistema operativo MS-DOS, vengono presentate le istruzioni del linguaggio Basic corredandole con molti esempi e viene descritto in dettaglio l'ambiente QuickBASIC che integra tutti gli strumenti necessari per la stesura, la messa a punto e l'esecuzione dei programmi.
Infine sono riportati vari esempi di programmazione di base allo scopo di mostrare l'uso pratico delle istruzioni Basic in precedenza esposte e la loro organizzazione nella struttura di un programma.

Marzia Bonecchi è coordinatore generale tecnico presso il Dipartimento di Matematica dell'Università degli Studi di Milano.

Luisa Zauli è assistente ordinario di Teoria e Applicazioni delle Macchine Calcolatrici presso il corso di laurea in Matematica dell'Università degli Studi di Milano.

ISBN 88-251-7068-8

BONECCHI, ZAULI
PROGRAMMARE IN QUICKBASIC
CittàStudiEdizioni

CittàStudiEdizioni

MARZIA
BONECCHI
LUISA
ZAULI

PROGRAMMARE IN QUICKBASIC

CittàStudiEdizioni

MARZIA BONECCHI
LUISA ZAULI

PROGRAMMARE IN QUICKBASIC

CittàStudiEdizioni

Introduzione

Il testo è nato principalmente dall'esigenza di fornire agli studenti del corso di laurea in Matematica dell'Università degli Studi di Milano, al loro primo impatto con gli elaboratori, un buon supporto didattico atto a renderli il più possibile autonomi nell'uso degli strumenti a loro disposizione.
Esso è quindi essenzialmente rivolto agli studenti dei corsi di "Laboratorio di Programmazione" e "Teoria ed Applicazione delle Macchine Calcolatrici" di detto corso di Laurea, che utilizzano per le esercitazioni i personal computers installati nel Laboratorio Didattico del Dipartimento di Matematica.
Può comunque essere utilizzato con profitto anche da tutti coloro, studenti e non, che vogliano impadronirsi velocemente del linguaggio Basic e dell'ambiente QuickBASIC.

Dopo una panoramica dei principali comandi del sistema operativo MS-DOS, vengono presentate le istruzioni del linguaggio Basic corredandole con molti esempi e viene descritto in dettaglio l'ambiente QuickBASIC che integra tutti gli strumenti necessari per la stesura, la messa a punto e l'esecuzione dei programmi.
Infine sono riportati vari esempi di programmazione di base allo scopo di mostrare l'uso pratico delle istruzioni Basic in precedenza esposte e la loro organizzazione nella struttura di un programma.
In appendice vengono descritte le modalità di utilizzo, i limiti e le possibilità del software ESERCIZI appositamente predisposto per il corso di "Laboratorio di Programmazione".

Per maggiori dettagli sugli argomenti esposti si possono consultare sia i manuali di utilizzo disponibili presso il Laboratorio Didattico sia i

seguenti testi presenti nella Biblioteca del Dipartimento di Matematica:

N. Balossino: "BASIC e applicazioni"
Lattes, 1985
T.C. Bartee: "Programmare in BASIC"
Zanichelli, 1983
M. Cicchetti: "Manuale di QUICKBASIC"
Jackson, 1990
B. Gottfried: "Programmare in BASIC"
Etas Libri, 1984
B. Petrillo: "BASIC programmazione strutturata e linguaggio"
Clup, 1984

Gli elaboratori a disposizione per le esercitazioni sono dei personal computers di diverso modello e lavorano sotto il sistema operativo MS-DOS di varie versioni dalla 3.30 alla 5.0.

Con il termine di *sistema operativo* (s.o.) si indica l'insieme dei programmi che gestiscono le risorse del calcolatore (memorie, periferiche, ecc.) per consentire all'utente di utilizzare il calcolatore stesso.

Ogni elaboratore è dotato di almeno 640 Kbytes di memoria dell'unità centrale (RAM), di disco fisso e di una o due unità a dischetto. Il video è monocromatico e grafico, con una risoluzione dipendente dalla scheda grafica in dotazione.

Ad ogni coppia di elaboratori è collegata una stampante ad aghi: un deviatore a selezione manuale ne permette l'uso da parte dell'uno o dell'altro elaboratore.

Le autrici ringraziano il prof. M. Lunelli per i preziosi suggerimenti e la dott. M. Cerri per l'aiuto prestato.

Milano, gennaio 1993

Convenzioni di scrittura

Nel testo sono usate le convenzioni tipografiche qui spiegate.

corsivo	Lo stile corsivo è usato nella definizione della sintassi di comandi e istruzioni per indicare le parti variabili, che l'utente deve sostituire di volta in volta con le informazioni opportune. È usato inoltre per evidenziare termini nuovi di cui si dà la definizione.
grassetto	Il grassetto è usato nella sintassi di comandi e istruzioni per le parole chiave, che l'utente deve riportare esattamente come sono (salvo poterle scrivere in minuscolo). È usato nella spiegazione di comandi e istruzioni per evidenziare gli stessi.
`typewriter`	Questo stile è usato per gli esempi di comandi e istruzioni, per l'input e l'output di programmi, per i messaggi d'errore.
Enter	In questo stile sono scritti i nomi dei tasti situati sulla tastiera del PC e sul pannello della stampante.
[*opzione*]	Nella sintassi di comandi e istruzioni le parti racchiuse tra parentesi quadre sono opzionali, cioè possono essere omesse. Le parentesi quadre non fanno parte del comando o istruzione.
{;|,}	Nella sintassi di comandi e istruzioni le parentesi graffe indicano una scelta obbligatoria tra le due o più alternative che compaiono separate dalla barra verticale.
[,*var*] ...	Nella sintassi di comandi e istruzioni tre puntini indicano l'eventuale ripetizione di più elementi uguali a quello che li precede.

Capitolo 1

Operazioni elementari per l'uso del PC

1.1 Accensione e spegnimento

Per utilizzare un elaboratore del Laboratorio Didattico del Dipartimento di Matematica è sufficiente accenderne l'unità centrale. Il video dovrebbe accendersi di conseguenza; in caso contrario premere il suo tasto d'accensione.
Attendere la comparsa sullo schermo del messaggio "`C:\USER>`". A questo punto l'elaboratore si trova in *ambiente MS-DOS*, cioè è pronto ad accettare comandi del sistema operativo.
Se si deve usare la stampante, assicurarsi che il deviatore sia posizionato opportunamente e che la carta sia allineata a inizio pagina.

Al termine del lavoro, lo studente è tenuto a cancellare tutte le informazioni da lui memorizzate su disco fisso e a spegnere l'unità centrale dell'elaboratore e la stampante.

1.2 Uso dei dischetti

I dischetti permettono agli utenti di conservare i propri programmi e dati.

Le unità a dischetto disponibili sono di tre tipi:

a) dimensione 5,25 pollici e capacità 360 Kbytes,

b) dimensione 5,25 pollici e capacità 1,2 Mbytes,

c) dimensione 3,5 pollici e capacità 1,44 Mbytes.

Sulle unità di tipo b) si possono leggere e scrivere anche dischetti di tipo a).
Sulle unità di tipo c) si possono leggere e scrivere anche dischetti di capacità 720 Kbytes.

Un dischetto nuovo deve essere predisposto per l'uso mediante il comando **FORMAT** (vedi oltre).

I dischetti da 5,25" presentano una piccola tacca rettangolare su uno dei bordi. Se la tacca è lasciata scoperta, si possono registrare nuovi dati sul disco; se invece la tacca viene coperta da un'apposita linguetta adesiva, il contenuto del disco può solo essere letto, non modificato o cancellato.

I dischetti da 3,5" presentano in un angolo un foro chiudibile mediante una linguetta a scorrimento. Il dischetto risulta protetto da scrittura quando il foro è aperto.

Il dischetto va inserito nell'apposita unità con l'etichetta rivolta verso l'alto e la tacca di protezione sul lato sinistro; i dischetti da 5,25" devono essere bloccati mediante l'apposito pulsante o levetta.

Attenzione!

Quando è in corso un'operazione di lettura o scrittura sul dischetto, la spia rossa dell'unità è accesa. Rimuovere il dischetto dall'unità solo quando la spia è spenta.
Non accendere o spegnere l'elaboratore con il dischetto inserito.

1.3 Uso della stampante

1.3.1 Pannello di controllo

Sulla stampante si trovano alcuni tasti e spie luminose.

Tasto *ON LINE*
funziona da interruttore per attivare o disattivare il collegamento della stampante con l'elaboratore; il tasto è dotato di una propria spia, che è accesa quando il collegamento è attivo.

Tasto *FORM FEED*
ha effetto solo quando la spia *ON LINE* è spenta; premendolo, si fa avanzare la carta fino all'inizio del foglio successivo.

Tasto *LINE FEED*
ha effetto solo quando la spia *ON LINE* è spenta; premendolo, si fa avanzare la carta di una riga; l'avanzamento continua fino a che il tasto rimane premuto.

Spia *POWER*
è illuminata quando la stampante è accesa.

Spia *READY*
si accende quando si attiva la stampante; durante la stampa lampeggia.

Spia *PAPER OUT*
si accende quando manca la carta.

Un altro insieme di tasti permette di scegliere lo stile di stampa, ovvero la fonte dei caratteri e la spaziatura tra di essi.
Le fonti più comuni sono:
DRAFT, che consente una stampa veloce ma di bassa qualità,
NLQ o *ROMAN* o *SANS SERIF*, che producono stampe di alta qualità ma a velocità ridotta.
Le spaziature possibili sono 10 o 12 pitch (caratteri per pollice) o la spaziatura *proporzionale* (dove ogni carattere occupa uno spazio proporzionale alla sua larghezza).
La stampa in compresso (*CONDENSED*) riduce la larghezza dei caratteri e permette la stampa di righe più lunghe di 80 caratteri.
Una opzione è attiva quando la spia corrispondente è illuminata.

1.3.2 Avanzamento della carta

Alcuni modelli di stampante dopo una stampa ottenuta con il comando **PRINT** di MS-DOS fanno avanzare automaticamente la carta per permetterne lo strappo (bisogna attende qualche secondo) e la riposizionano all'atto della stampa successiva.
Se la stampante si comporta diversamente, per fare avanzare la carta dopo una stampa occorre eseguire le seguenti operazioni:

a) disattivare la stampante premendo il tasto *ON LINE* (si spegne la spia);

b) far avanzare la carta con i tasti *FORM FEED* (avanzamento a pagina nuova) o *LINE FEED* (avanzamento di una riga);

c) riattivare la stampante premendo nuovamente il tasto *ON LINE*.

Non utilizzare mai la manopola a stampante accesa. Infatti la stampante ha un meccanismo automatico di conteggio delle righe e di allineamento a pagina nuova; usando la manopola a stampante accesa si ottiene solo di alterare questo conteggio e si rischia anche di rompere la stampante.

Fare attenzione che la levetta di selezione del tipo di carta (modulo continuo o foglio singolo) sia sempre posizionata sul modulo continuo. In caso contrario la carta si inceppa perché scorre male.

1.4 Uso della tastiera

Il funzionamento della tastiera dipende dal sistema operativo installato sul calcolatore e dal programma che si sta usando.

Qui di seguito sono riportate le funzioni dei tasti speciali più importanti in MS-DOS.

Ctrl ci sono due tasti *Ctrl* che possono essere usati indifferentemente. *Ctrl* da solo non ha nessun effetto; se, tenendolo premuto, si batte un altro carattere, si compie una

funzione che dipende dal carattere battuto. Le combinazioni più utili sono riportate qui di seguito.

Ctrl C interrompe il comando o il programma in corso di esecuzione.

Ctrl Break svolge funzioni identiche a *Ctrl C* (Il tasto *Break* riporta sulla parte superiore la dicitura *Pause*).

Ctrl S sospende lo scorrimento del testo emesso su video; premendo un tasto qualsiasi la visualizzazione riprende.

Ctrl P attiva o disattiva la riproduzione su stampante di tutto ciò che compare via via sul video; normalmente la stampa non viene effettuata.

Ctrl Alt Del ricarica il sistema operativo (prima ricordarsi di togliere il dischetto!). Equivale quasi a spegnere e riaccendere il calcolatore ma di solito è preferibile. Tutto il contenuto della memoria viene azzerato.

Caps Lock premendolo (spia corrispondente accesa) si fissa il maiuscolo per le lettere da A a Z; premendolo ancora si ripristinano le minuscole.

Shift ci sono due tasti *Shift* che possono essere usati indifferentemente. Tenendo premuto *Shift* mentre si batte un carattere alfabetico, questo viene riprodotto su video in maiuscolo se *Caps Lock* non è attivo, altrimenti in minuscolo. Tenendo premuto *Shift* mentre si batte un tasto contrassegnato da due caratteri, si ottiene il carattere superiore.

Pause svolge funzioni identiche a *Ctrl S*.

Print Screen copia su stampante il contenuto della pagina video.

Backspace cancella il carattere alla sinistra del cursore.

Enter termina sempre una riga battuta da tastiera.

Nella parte destra della tastiera si trovano 10 tasti (denominati *Insert*, *Delete*, *Home*, *End*, *Page Up*, *Page Down*, ←, →, ↑, ↓), che controllano l'inserimento e la cancellazione di caratteri e i movimenti del cursore. Il loro uso dipende dall'ambiente di lavoro; successivamente ne verrà spiegato il significato nell'ambiente QuickBASIC.

Accanto a questi tasti compare una piccola tastiera, che chiameremo

tastierino numerico, che può servire per due scopi:

- per produrre le cifre 0, 1, ..., 9, gli operatori *, -, +, / e il punto decimale;
- per controllare i movimenti del cursore e l'inserimento e la cancellazione di caratteri, in alternativa ai 10 tasti precedenti.

I tasti servono per il primo scopo quando la spia *Num Lock* è accesa; premendo il tasto *Num Lock* la spia si spegne e si attivano le funzioni del secondo tipo, fino alla nuova pressione dello stesso tasto.

Nella parte alta della tastiera si trovano 12 tasti funzione *F1*, *F2*, ..., *F12*, che hanno anch'essi un uso dipendente dall'ambiente di lavoro. Successivamente per alcuni di essi verrà dato il significato in ambiente MS-DOS e QuickBASIC.

Capitolo 2

Unità periferiche e files in MS-DOS

2.1 Nomi delle unità periferiche

Il sistema assegna alle unità periferiche i seguenti nomi:

C:	unità a disco fisso
A:	unità a dischetto
B:	eventuale seconda unità a dischetto
CON:	console (tastiera+video)
PRN: o LPT1:	porta parallela (stampante)
COM1:	porta seriale

Il carattere due punti deve sempre seguire il nome.

2.2 Organizzazione dei dati su disco

I dati ed i programmi che vengono utilizzati sono memorizzati su disco fisso o dischetto in *archivi* o *files* organizzati in diversi *direttori* secondo una struttura gerarchica ad albero. Un *direttorio* è un elenco di nomi di files e/o di altri direttori.

Dal direttorio *radice* dell'albero si diramano files di dati e programmi oppure nomi di direttori che a loro volta possono contenere altri files ed altri direttori.

Nella struttura ad albero i direttori ne costituiscono quindi i rami e i files le foglie. Il direttorio radice costituisce il primo livello della struttura.

Un'organizzazione di questo tipo permette di suddividere i files in gruppi omogenei rispondenti alle necessità dell'utente.

La figura seguente illustra una tipica struttura ad albero di files e direttori sul dischetto A:
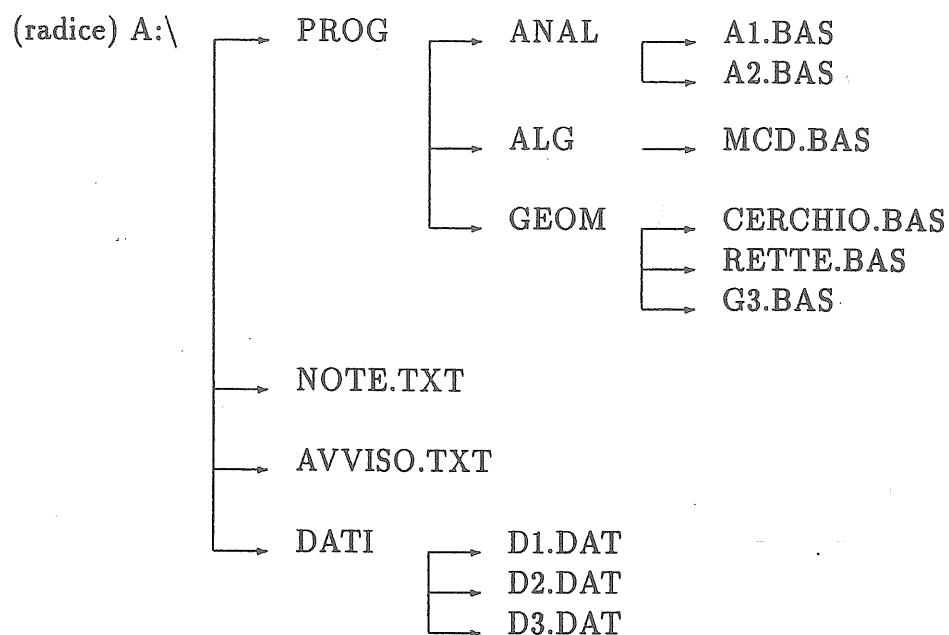
Figura 2.1

Il direttorio radice contiene i due files NOTE.TXT e AVVISO.TXT e i due direttori subordinati PROG e DATI.

A sua volta DATI contiene i tre files di dati D1.DAT, D2.DAT e D3.DAT, mentre PROG è ulteriormente strutturato nei tre direttori di livello inferiore ANAL, che contiene due files di programmi Basic, ALG che ne contiene uno e GEOM che ne contiene tre.

Tutti gli esempi successivi faranno riferimento a questa struttura ad albero.

I nomi dei direttori e dei files sottostanno alle regole fornite nel seguito.

2.3 Nomi dei direttori

Il direttorio radice ha nome standard *d*:\ dove *d* è il nome dell'unità disco.

Esempi:

C:\	direttorio radice del disco fisso
A:\	direttorio radice del dischetto nell'unità A
B:\	direttorio radice del dischetto nell'unità B

Un nome di direttorio è costituito al massimo di 8 caratteri alfabetici o numerici.

Esempi:
PROG
p123
ALG
1234

2.4 Nomi dei files

Ogni file è individuato da un nome, in cui si distinguono due parti separate da un punto.

La prima parte è il *nome* vero e proprio, costituito al massimo di 8 caratteri alfabetici o numerici.

La seconda parte, o *estensione*, lunga al massimo 3 caratteri, è opzionale e serve a indicare il tipo di dati contenuti nel file. Ad esempio files di programmi sorgente scritti in Basic hanno estensione BAS.

Esempi:
CERCHIO.BAS
MCD.BAS

L'estensione è vivamente raccomandata. Per i dati è opportuno usare l'estensione DAT.

Esempio:
D1.DAT

I seguenti nomi non possono essere usati come nomi di file, perché MS-DOS associa ad essi un significato speciale (normalmente i nomi delle periferiche):

AUX, CON, LPT1, PRN, NUL, COM1.

2.5 Uso dei caratteri jolly

Se si deve eseguire una stessa operazione su più files che hanno la parte iniziale del nome o dell'estensione in comune, è possibile dare un unico comando, valido per tutti i files, usando il carattere jolly "*". Esso sostituisce un qualunque gruppo di caratteri, anche l'intero nome o l'intera estensione.

Esempi:

*.BAS	individua tutti i files contenenti programmi sorgente in Basic;
PROVA.*	individua tutti i files con nome PROVA ed estensione qualunque;
.	individua tutti i files;
F*.BAS	individua tutti i files contenenti programmi in Basic il cui nome inizi per F.

2.6 Pathname (percorso)

Per operare su un file o su un direttorio bisogna individuarne la posizione all'interno della struttura attraverso il suo *pathname*, cioè il percorso che congiunge il direttorio radice al file o direttorio in questione.

Tale percorso è individuato dalla sequenza dei nomi dei direttori via via toccati a partire dalla radice fino al file; la sequenza si conclude

col nome del file. I nomi sono separati l'uno dall'altro da una barra rovesciata che individua il passaggio da un nodo al nodo *figlio* nella struttura dei dati.

Esempi:

A:\PROG\ANAL\A1.BAS   individua il file A1.BAS nella struttura di fig. 2.1.

A:\DATI   individua il direttorio DATI nella struttura di fig. 2.1.

Poiché il percorso individua un file in modo univoco, files o direttori con lo stesso nome possono esistere in direttori diversi.

2.7 Disco attivo e direttorio di lavoro

In realtà non è sempre necessario scrivere un pathname per esteso. Il nome del disco può essere omesso se si tratta del disco *attivo* al momento.

All'accensione del PC il disco attivo è C. Un solo disco può essere attivo in ogni istante.

In ogni momento l'utente si trova inoltre *posizionato* in un particolare direttorio della struttura ad albero del disco attivo (*direttorio di lavoro*).

I files contenuti nel direttorio di lavoro possono essere individuati semplicemente con nome ed estensione.

I files e i direttori a livelli successivi si possono individuare con un pathname ridotto, in cui si tralascia la parte dalla radice al direttorio di lavoro.

Esempi:

Se il disco attivo è A e il direttorio di lavoro è DATI, il pathname A:\DATI\D1.DAT può essere abbreviato in D1.DAT.

Se il direttorio di lavoro è PROG, le seguenti coppie di pathname sono equivalenti:

A:\PROG\ALG e ALG.
A:\PROG\GEOM\G3.BAS e GEOM\G3.BAS.

Per i PC del Laboratorio Didattico, al momento dell'accensione il direttorio di lavoro è C:\USER.

Capitolo 3

Principali comandi del sistema operativo MS-DOS

In ogni istante è possibile impartire all'elaboratore solo i comandi riconosciuti dal programma che in quel momento ne ha il *controllo*. Ad esempio comandi Basic possono essere impartiti solo dopo aver attivato un programma interprete del linguaggio Basic. È indispensabile quindi conoscere l'*ambiente di lavoro*.

I comandi elencati in questo capitolo sono presenti in tutte le versioni del sistema operativo MS-DOS installate nel Laboratorio. Tali comandi possono essere dati solo in ambiente MS-DOS, cioè quando sul video compare il pathname del direttorio di lavoro seguito dal carattere ">". Questo messaggio è il cosiddetto *prompt* del sistema.

Esempi di prompt:

```
C:\USER>
A:\>
A:\PROG\ANAL>
```

Tutti i comandi vanno conclusi premendo il tasto *Enter*, che provoca la loro esecuzione.

I comandi possono essere scritti sia in maiuscolo che in minuscolo.

Per ripetere più volte consecutive l'esecuzione di uno stesso comando, basta scrivere il comando la prima volta e dopo l'esecuzione richiamarlo premendo il tasto *F3*.

3.1 Preparazione di un dischetto nuovo per l'uso

I dischetti nuovi vanno predisposti all'uso con l'operazione di *formattazione* mediante il comando **FORMAT**.
Se il dischetto non è nuovo, la formattazione ne cancella tutto il contenuto; quindi il comando va usato con cautela.

Il comando crea il *direttorio radice* e la FAT (*File allocation table*), che tiene traccia dell'occupazione del disco.

La sintassi del comando per la formattazione di un dischetto posto nell'unità *d* è:

FORMAT *d*: [*opzioni*]

Tra le opzioni, facoltative, le più usate sono:

/V	permette di attribuire un nome identificativo al dischetto al termine della formattazione;
/4	formatta un dischetto da 360 Kbytes 5,25" in una unità da 1,2 Mbytes;
/N:9 /T:80	formatta un dischetto da 720 Kbytes 3,25" in una unità da 1,44 Mbytes.

Se la formattazione è fatta su una unità avente le stesse caratteristiche del dischetto non è necessaria nessuna opzione che le definisca.

Esempi:
`FORMAT A:`
`FORMAT B: /4 /V`

L'elaboratore chiede di inserire il dischetto nell'unità (A o B) indicata e di battere *Enter*.
Nelle versioni di MS-DOS più recenti, alla fine della formattazione viene comunque richiesto il nome da assegnare al dischetto (*etichetta del volume*), anche se non è stata specificata l'opzione /V.
Viene infine richiesto se si devono formattare altri dischetti: rispondere *S* per sì o *N* per no.

3.2 Duplicazione di un intero dischetto

Il comando

DISKCOPY *d1*: *d2*:

copia tutto il contenuto del dischetto inserito nell'unità *d1* sul dischetto inserito nell'unità *d2*.

Le due unità devono essere dello stesso tipo, perciò generalmente esse coincidono e MS-DOS guida l'avvicendamento dei due dischetti chiedendo di inserire nell'unità prima il dischetto *origine* e poi quello *destinazione*.

Esempio:
`DISKCOPY B: B:`

Non è necessario formattare prima il dischetto destinazione.

Anche per la duplicazione, come per la formattazione, l'operazione può essere eseguita su più dischetti successivamente.

Per copiare integralmente un dischetto di un tipo su uno di tipo diverso si devono usare i comandi **COPY** o **XCOPY** descritti nel seguito.

3.3 Cambio del disco attivo

Per posizionarsi su una unità disco diversa dall'attuale basta scrivere il nome del disco che si vuole rendere attivo.

Esempi:
`A:` attiva il dischetto nell'unità A
`C:` attiva il disco fisso

3.4 Operazioni sui direttori

Nel seguito con *nome-dir* e *nome-file* si intenderanno rispettivamente il pathname, completo o ridotto, del direttorio o del file desiderato.
Gli esempi sono riferiti alla struttura di figura 2.1.

3.4.1 Lista del contenuto di un direttorio

Il comando

DIR

elenca i nomi dei files e dei direttori contenuti nel direttorio di lavoro con data ed ora di creazione.

DIR *nome-dir*

ha lo stesso significato ma per il direttorio indicato.

DIR *nome-file*

visualizza solo la parte di direttorio relativa al file indicato (si usa di solito con il carattere "*").

Si possono aggiungere al comando le seguenti opzioni o specifiche:

/P	interrompe la visualizzazione della lista alla fine di ogni pagina video;
/W	visualizza in modo compatto (5 per riga) solo i nomi;

Nella lista del contenuto di un direttorio che non sia la radice, le prime due voci sono sempre:

.	che individua il direttorio stesso;
..	che individua il direttorio immediatamente precedente nell'albero (direttorio *padre*).

Esempi:

`DIR GEOM /W`
lista (in modo compatto) il contenuto di GEOM se il direttorio di lavoro è A:\PROG.

`DIR A:\PROG\GEOM /W`
come il precedente ma qualunque sia il direttorio di lavoro.

`DIR *.BAS`
elenca tutti i files di programmi Basic presenti nel direttorio di lavoro.

3.4.2 Creazione di un direttorio

Il direttorio radice del disco è generato dal comando FORMAT.

Un qualunque altro direttorio può essere creato con il comando

MD *nome-dir*

Il direttorio padre di *nome-dir* deve già esistere.

Esempi:

`MD ALG`
crea ALG se il direttorio di lavoro è A:\PROG.

`MD A:\PROG\ALG`
crea ALG qualunque sia il direttorio di lavoro.

3.4.3 Cambio del direttorio di lavoro sul disco attivo

Il posizionamento in un determinato direttorio si ottiene con il comando

CD *nome-dir*

Esempi:

`CD ALG`
cambia il direttorio di lavoro da A:\PROG a A:\PROG\ALG.

`CD A:\PROG\ALG`
cambia da qualsiasi direttorio del dischetto A a A:\PROG\ALG.

`CD ..`
riporta al direttorio immediatamente precedente (direttorio padre).

`CD \`
riporta al direttorio radice del disco attivo.

3.4.4 Eliminazione di un direttorio

Per cancellare un direttorio si usa il comando

RD *nome-dir*

Il direttorio deve essere vuoto, cioè devono essere stati cancellati tutti i files e i direttori in esso contenuti.
Ad esempio, se il direttorio di lavoro è A:\PROG, per eliminare ALG bisogna dare i seguenti comandi (il comando DEL è spiegato in 3.5.3):
`DEL ALG\*.*` (basta anche `DEL ALG`)
`RD ALG`

3.5 Operazioni sui files

Anche in questa sezione gli esempi si riferiscono alla figura 2.1.

3.5.1 Visualizzazione su video del contenuto di un file

Il comando

TYPE *nome-file*

visualizza sullo schermo il contenuto di un file.

3.5.2 Lista su stampante del contenuto di un file

Il comando

PRINT *nome-file*

stampa sulla carta il contenuto di un file.

Se compare la richiesta `Nome della periferica di stampa [PRN]:` premere il tasto *Enter*.

Il comando ammette l'uso del carattere jolly "*" per stampare fino a 10 files aventi nomi con parti in comune.

3.5.3 Cancellazione di un file

Per cancellare un file si usa il comando

DEL *nome-file*

Esso ammette l'uso del carattere jolly "*" per eliminare più files aventi nomi con parti in comune.

Esempio:
`DEL F*.BAS`
cancella tutti i programmi Basic il cui nome inizia con F.

3.5.4 Cambio del nome di un file

Il comando

REN *nome-vecchio nome-nuovo*

cambia nome al file *nome-vecchio* attribuendogli il nome *nome-nuovo*.

Mentre *nome-vecchio* può essere un pathname completo, *nome-nuovo* deve specificare solo nome ed estensione nuovi da attribuire al file; pertanto non è possibile usare il comando REN per spostare un file su un altro disco o in un altro direttorio.

Esempi:

`REN D1.DAT DATI.DAT`
se il direttorio di lavoro è A:\DATI, cambia il nome del file D1.DAT in DATI.DAT.

`REN A:\PROG\ANAL\A1.BAS A3.BAS`
qualunque sia il direttorio di lavoro, cambia il nome del file A1.BAS in A3.BAS nel direttorio A:\PROG\ANAL.

Il comando ammette l'uso del carattere jolly "*". Ad esempio, per cambiare tutte le estensioni DAT in D nel direttorio di lavoro:
`REN *.DAT *.D`

3.5.5 Copia di un file

Il comando

COPY *nome-file nome-copia* [/V]

genera una copia del file *nome-file* e attribuisce al nuovo file il pathname *nome-copia*. Il comando consente di fare la copia di uno o più files; la copia può essere generata nello stesso direttorio (con un altro nome), in un altro direttorio o anche su un altro disco. L'opzione /V effettua la verifica della copia eseguita.

Nel pathname *nome-copia* l'ultima parte (nome ed estensione del file) può essere tralasciata se uguale a quella di partenza.

Esempi:

(Il direttorio di lavoro sia A:\ ossia la radice del dischetto.)

`COPY NOTE.TXT NOTE1.TXT`
genera nello stesso direttorio una seconda copia del file NOTE.TXT avente nome NOTE1.TXT.

`COPY NOTE.TXT PROG\ALG\NOTE1.TXT`
`COPY NOTE.TXT PROG\ALG`
generano nel direttorio A:\PROG\ALG la copia di NOTE.TXT, con il nome NOTE1.TXT nel primo caso e NOTE.TXT (che si può omettere nel comando) nel secondo.

`COPY NOTE.TXT C:\USER`
copia NOTE.TXT dal dischetto al direttorio C:\USER del disco fisso.

`COPY A:\*.BAS B:\`
copia tutti i files Basic dal direttorio radice del dischetto nell'unità A al direttorio radice del dischetto in B. Se il PC in dotazione ha un'unica unità a dischetto, il sistema guiderà l'avvicendamento dei due dischetti nella stessa unità chiedendo di volta in volta il dischetto *sorgente* (A) e quello *destinazione* (B) della copia.

3.5.6 Copia di files e/o direttori

Il comando XCOPY ha due varianti:

XCOPY *nome-file* *nome-copia* [/V] [/D:*data*]

XCOPY *nome-dir* *nome-dir-copia* [/V] [/S] [/D:*data*]

Nella prima forma XCOPY equivale a COPY, salvo che permette di selezionare, mediante l'opzione /D, solo i files con data eguale o posteriore a quella indicata.

Esempio:
`XCOPY A:\PROG\GEOM\*.BAS B:\ /D:20-02-92`

Nella seconda forma, il comando copia tutti i files del direttorio *nome-dir* nel direttorio *nome-dir-copia*, mantenendo gli stessi nomi.
Se è presente l'opzione /S vengono ricopiati anche tutti i direttori sottostanti a *nome-dir*, ricostruendo la stessa struttura sotto *nome-dir-copia*.

Esempio:
`XCOPY A:\ B:\ /S /V`
copia tutta la struttura dei files del dischetto A sul dischetto B effettuando la verifica.

3.5.7 Concatenazione di files

Il comando COPY consente anche di unire più files in uno solo accodandoli l'uno all'altro.

Esempi:

`COPY A:\PROG\GEOM\A*.BAS C:\USER\GEOMETRIA`
copia nell'unico file GEOMETRIA del direttorio USER del disco fisso tutti i programmi Basic situati nel direttorio A:\PROG\GEOM e aventi nome che inizia con la lettera A.

`COPY A1.BAS+A2.BAS A3.BAS`
se il direttorio di lavoro è A:\PROG\ANAL, concatena nell'unico file A3.BAS i due files A1.BAS e A2.BAS.

3.6 Controllo del flusso dei dati

Alcuni comandi del s.o. richiedono informazioni aggiuntive (*input*) e quasi tutti inviano risultati e messaggi (*output*).

Le unità standard di input e output sono rispettivamente la tastiera ed il video.
Questa assegnazione standard può essere temporaneamente modificata tramite la *ridirezione* che si effettua mediante i seguenti simboli:

>*nome*
l'output viene inviato sul file o periferica individuata da *nome*.

<*nome*
l'input viene preso dal file *nome*.

>>*nome*
l'output viene accodato al file *nome*.

Il controllo del flusso di dati può avvenire mediante i *filtri* che sono comandi particolari che trasformano i dati secondo opportune modalità. Essi richiedono che il disco attivo non sia protetto da scrittura e usano come unità standard la tastiera e lo schermo. Per modificare le unità standard si usano i simboli di *ridirezione*.

Il filtro MORE presenta l'output su schermo una pagina per volta. Tra una pagina e l'altra compare in fondo allo schermo il messaggio *More* o *Continua* a cui bisogna dare *Enter* per passare alla schermata successiva o *Ctrl Break* per interrompere.

Il filtro SORT riordina alfabeticamente l'input.

I comandi filtro possono ricevere l'input da un altro comando utilizzando il carattere "|" (*pipe*) per separare i comandi sulla riga.

Gli esempi che seguono chiariranno meglio l'uso di tali comandi:

`DIR >PRN`
lista il direttorio sulla stampante (questa specifica deve comparire in fondo al comando)

`DIR >>nome`
accoda la lista del direttorio al file `nome`

`DIR | sort >nome`
lista il direttorio in ordine alfabetico sul file `nome`

`<old SORT >new`
ordina i dati presi dal file `old` ponendoli nel file `new`

`MORE <new`
visualizza sullo schermo il file `new` una pagina per volta

`TYPE old |MORE`
visualizza sullo schermo il file `old` una pagina per volta

Capitolo 4

Elementi principali del linguaggio Basic

Un *programma* in linguaggio Basic si compone di una serie di *istruzioni*, ciascuna delle quali ordina all'elaboratore di svolgere una specifica operazione.

Come per tutti i linguaggi ad alto livello, le istruzioni Basic (che costituiscono il programma *sorgente*) per poter essere eseguite devono essere tradotte in *linguaggio macchina*, cioè nel linguaggio binario comprensibile all'elaboratore; questo compito è svolto da opportuni programmi di utilità detti *traduttori*.

I tipi "classici" di traduttori di linguaggio sono:
- l'*interprete*, che decodifica ed esegue immediatamente ogni istruzione del programma sorgente prima di passare alla successiva;
- il *compilatore*, che traduce tutto il programma sorgente in linguaggio macchina, generando un programma *oggetto* la cui esecuzione potrà avvenire in un secondo tempo.

La modalità interpretata permette un uso interattivo che facilita la preparazione e la prova dei programmi; la modalità compilata consente di ottenere, con poche modifiche al sorgente, programmi di più veloce esecuzione.

In questo testo si parlerà dell'ambiente QuickBASIC, che riunisce in sé i vantaggi di entrambi i tipi di traduttori.

Ogni linea di un programma Basic può contenere una o più istruzioni separate l'una dall'altra dal carattere ":" e può essere preceduta da una *etichetta* o *label* che la identifica. Può anche essere conclusa da un *commento*, come si vedrà in 5.1. La lunghezza massima è di 256 caratteri. Il formato di una linea è dunque

[*etichetta*] *istruzione* [*:istruzione*] ... [*commento*]

Gli elementi principali che compaiono in una istruzione sono parole chiave, etichette, dati (costanti, variabili semplici e con indice) ed operatori.

4.1 Parole chiave

Le *parole chiave* sono parole caratteristiche del linguaggio e individuano il tipo di operazione che l'istruzione deve svolgere.
Nel seguito verranno sempre indicate in maiuscolo, anche se l'utente può scriverle in minuscolo.

Le parole chiave sono riservate, cioè non possono essere usate come nomi di altri elementi di una frase (etichette o variabili).

4.2 Etichette

Le *etichette* possono essere numeriche (un intero da 1 a 65529) o alfanumeriche (fino a 40 caratteri maiuscoli o minuscoli, di cui il primo alfabetico); nel secondo caso devono terminare con il carattere ":".

Esempi di etichette valide:
`20` `a12:` `abc:`

La presenza di un'etichetta è obbligatoria solo sulle linee che vengono richiamate da altri punti del programma. Pertanto i numeri di linea non determinano l'ordine in cui le istruzioni vengono eseguite.

4.3 Tipi di dati

I dati possono essere *costanti*, cioè valori che non subiscono modifiche durante l'esecuzione di un programma, o *variabili*, cioè nomi che individuano celle di memoria contenenti valori che cambiano nel corso del calcolo.

Ci sono due categorie di dati: i dati di tipo stringa e quelli di tipo numerico. Il tipo caratterizza il modo con cui il dato è rappresentato nella memoria dell'elaboratore.

I dati di tipo *stringa* sono sequenze di caratteri, ciascuno dei quali occupa un byte di memoria. Possono essere costituiti da un numero di caratteri non superiore a 32767.
Ogni carattere è rappresentato da una codifica numerica, secondo la cosiddetta *rappresentazione ASCII*. Per esempio il carattere "A" ha per codice ASCII il valore decimale 65.

I dati *numerici* possono essere *interi* o in *virgola mobile* (questi ultimi sono chiamati impropriamente *reali*), in *semplice* o in *doppia precisione*.
Gli interi in semplice precisione (chiamati semplicemente *interi*) occupano 2 bytes di memoria, gli interi in doppia precisione (*interi lunghi*) ed i reali in semplice occupano 4 bytes, i reali in doppia precisione ne occupano 8.
La tabella seguente riporta l'intervallo dei valori rappresentabili per ciascun tipo di dato:

interi	$-32768 \le x \le 32767$
interi lunghi	$-2147483648 \le x \le 2147483647$
reali in s. p.	$-3.402823 \times 10^{38} \le x \le -1.40129 \times 10^{-45}$
	$x = 0$
	$1.40129 \times 10^{-45} \le x \le 3.402823 \times 10^{38}$
reali in d. p.	$-1.797693134862315 \times 10^{308} \le x \le -4.94065 \times 10^{-324}$
	$x = 0$
	$4.94065 \times 10^{-324} \le x \le 1.797693134862315 \times 10^{308}$

Tutti i numeri sono rappresentati con un numero finito di cifre. Ne consegue che gli interi compresi negli intervalli riportati nella tabella

sono tutti rappresentati esattamente nell'elaboratore, mentre ogni reale compreso nell'intervallo di rappresentazione è approssimato da un numero in virgola mobile con un numero finito di cifre significative.
I numeri in virgola mobile in precisione semplice hanno 7 cifre decimali significative, quelli in precisione doppia ne hanno 14.

4.3.1 Costanti

Le costanti sono dati che si utilizzano nelle espressioni ed hanno un valore predefinito.

1. *Costanti stringa*:

   sono costituite da una sequenza di caratteri racchiusa tra doppi apici.

   Esempi:

   `"MILANO"` `"Personal Computer"` `"1+2+3="`

2. *Costanti numeriche*:

   intere: sono numeri interi relativi compresi nell'intervallo di rappresentazione degli interi lunghi. A seconda del valore, vengono assegnati a tali costanti due o quattro bytes.

   Esempi:

   `407` (intero)

   `-33000` (intero lungo)

   `+2377401` (intero lungo)

   in virgola mobile: sono numeri in forma decimale o esponenziale; nel secondo caso il numero decimale è immediatamente seguito dalla lettera E (semplice precisione) o D (doppia precisione), che individua implicitamente la base 10, e da un esponente con segno.

   Esempi:

   `235.9` `235E-7` `-235.98D6`

4.3.2 Variabili semplici

Le variabili semplici sono identificatori che individuano un singolo dato, di tipo numerico o stringa, il cui valore può cambiare nel corso dell'esecuzione di un programma.

I nomi delle variabili devono iniziare con una lettera, possono contenere fino a 40 caratteri alfanumerici e sono a scelta dell'utente.

Le variabili di cui non viene specificato esplicitamente il tipo in uno dei modi sottoriportati sono considerate numeriche in virgola mobile ed in semplice precisione.

Il tipo può essere specificato in tre modi:

- mediante l'aggiunta di un carattere in fondo al nome:

%	intero	Es.: `alfa%`, `d1%`
&	intero lungo	`d39&`, `beta&`
!	reale in semplice precisione	`rad!`
#	reale in doppia precisione	`pigre#`
$	stringa	`nome$`

- mediante l'istruzione

  DIM *var* AS *tipo* [,*var* AS *tipo*] ...

  ove *var* è l'identificatore di una variabile semplice e *tipo* può essere **INTEGER, LONG, SINGLE, DOUBLE** o **STRING.**

  Esempio:

  ```
  DIM b AS INTEGER, somma AS DOUBLE
  ```

- usando le istruzioni dichiarative

DEFINT	per gli interi
DEFLNG	per gli interi lunghi
DEFSNG	per i reali in semplice precisione
DEFDBL	per i reali in doppia precisione
DEFSTR	per le stringhe

Per cancellare dalla memoria una o più tabelle dinamiche si usa l'istruzione

ERASE *nome* [,*nome*] ...

Nel caso di tabelle statiche l'istruzione ERASE reinizializza gli elementi.

Le dimensioni di una tabella dinamica (ma non il loro numero) possono essere cambiate mediante l'istruzione REDIM che ha la stessa sintassi della DIM e reinizializza gli elementi.

4.3.4 Tipi di dati definiti dall'utente

L'utente ha la possibilità di definire, mediante l'istruzione TYPE, una propria struttura di dati comprendente elementi di tipo diverso. Successivamente può utilizzare variabili con questa struttura dopo averle dichiarate tali.

La struttura viene così definita:

TYPE *nometipo*
el1 AS *tipo*
[*el2* AS *tipo*]
... ...
END TYPE

nometipo è il nome assegnato al nuovo tipo di dato;

el1, *el2*, ... sono nomi di variabili semplici;

tipo è il tipo di ciascuna variabile e può essere: INTEGER, LONG, SINGLE o DOUBLE per le variabili numeriche; STRING*n per le stringhe, ove n è la lunghezza fissa della stringa.

Per dichiarare una variabile avente questa struttura basta usare l'istruzione

DIM *var* AS *nometipo*

I singoli elementi della struttura sono individuati dai nomi *var.el1*, *var.el2*, ...

Esempio:
la porzione di programma che segue definisce un nuovo tipo di dato, la struttura `complex`, composta da una coppia di variabili reali rappresentanti la parte reale e quella immaginaria di un numero complesso, e dichiara di tipo `complex` tre variabili `x`, `y` e `p`.

```
TYPE complex
  re AS SINGLE
  im AS SINGLE
END TYPE
DIM x AS complex, y AS complex, p AS complex
... ...
```

Dopo queste definizioni, le parti reali di `x`, `y` e `z` sono individuate rispettivamente da `x.re`, `y.re` e `p.re` e le parti immaginarie da `x.im`, `y.im` e `p.im`.

4.4 Espressioni ed operatori

Una *espressione* può essere costituita da una costante, da una variabile o da una combinazione di costanti e variabili legate da *operatori*.
Gli operatori Basic possono essere di diverso tipo: aritmetici, di confronto, logici, di stringa e funzionali.

4.4.1 Operatori aritmetici

Gli operatori *aritmetici* sono i seguenti, in ordine di precedenza:

^	elevamento a potenza	x^y
-	cambiamento di segno	-x
* /	moltiplicazione e divisione in virgola mobile	x*y x/y
\	divisione "intera"	x\y
MOD	resto della divisione "intera"	x MOD y
+ -	addizione e sottrazione	x+y x-y

Nella divisione "intera" gli operandi sono arrotondati all'intero più vicino, viene eseguita la divisione e come quoziente viene presa la parte intera del risultato.

L'ordine di esecuzione delle operazioni può essere modificato, come nell'algebra usuale, mediante l'uso delle parentesi.

Per esempio, l'espressione algebrica $\frac{x+y}{z}$ si scrive in Basic `(x+y)/z`.

4.4.2 Operatori di confronto

Gli operatori *di confronto* permettono di confrontare due valori; il risultato del confronto è *vero* (uguale a -1) o *falso* (uguale a 0) e può essere usato per modificare eventualmente il flusso del programma. Sono i seguenti:

$=$	uguale	$x = y$
$<>$ (o $><$)	diverso	$x <> y$
$<$	minore	$x < y$
$>$	maggiore	$x > y$
$<=$	minore o uguale	$x <= y$
$>=$	maggiore o uguale	$x >= y$

Gli operatori aritmetici hanno la precedenza su quelli di confronto. Per esempio l'espressione `4+2>3` ha valore -1 (vero).

Tramite gli operatori di confronto si possono paragonare anche le stringhe.
I confronti vengono fatti carattere per carattere, tra i codici di macchina (codici ASCII) dei caratteri corrispondenti, fino a che si trova una differenza o una delle due stringhe termina; per le lettere vale l'ordine alfabetico, le maiuscole precedono le minuscole, i caratteri numerici precedono le lettere, lo spazio li precede entrambi.
Si ha quindi il seguente ordinamento dei caratteri: spazio$<0<1<2<\ldots<9<A<B<C<\ldots<Z<a<b<\ldots<z$.

Le seguenti relazioni sono tutte vere:

```
"ALFA" < "BETA"
"ALFA" < "alfa"
"CON" < "CONO"
"ALF1" < "ALFA"
"AL FA" < "ALFA"
"1234" < "56"
```


4.4.3 Operatori logici

Gli operatori *logici* usano l'aritmetica booleana per definire una connessione logica tra i risultati di operazioni di confronto. In ordine di precedenza sono: NOT, AND, OR, XOR. Qui di seguito ne sono riportate le tavole di verità, ove V sta per vero e F per falso.

x	y	NOT x	x AND y	x OR y	x XOR y
V	V	F	V	V	F
V	F	F	F	V	V
F	V	V	F	V	V
F	F	V	F	F	F

Ad esempio `d<20 AND f<4` è vero solo se `d` è minore di 20 e contemporaneamente `f` minore di 4.

4.4.4 Operatori di stringa

Le stringhe possono essere concatenate tra loro mediante il segno di addizione (+).

Esempi:

`"FILE"+"NAME"` ha per risultato `"FILENAME"`.

Se `a$` contiene `"carta"` e `b$` contiene `"pesta"` l'operazione `a$+b$` accosta le stringhe `a$` e `b$` dando come risultato la stringa `"cartapesta"`.

4.4.5 Operatori funzionali

Una *funzione* può essere considerata un algoritmo che fornisce un unico risultato. È definita mediante il nome, seguito da una lista di variabili (*parametri* o *argomenti formali*) poste tra parentesi e separate tra loro da virgole.

Per usare la funzione, è sufficiente che il suo nome compaia in un'espressione seguito tra parentesi dalla lista degli operandi effettivi (*argomenti attuali*) su cui il calcolo deve essere eseguito.
Mentre gli argomenti formali sono sempre nomi di variabili semplici, gli argomenti attuali possono essere variabili semplici o con indice od

LCASE$($x$$)	converte la stringa $x$$ in minuscolo.
UCASE$($x$$)	converte la stringa $x$$ in maiuscolo.

Capitolo 5

Istruzioni del linguaggio Basic

Le istruzioni del linguaggio Basic si suddividono in *eseguibili* e *non eseguibili*.

Le prime sono istruzioni che fanno avanzare il flusso logico del programma specificando all'elaboratore le operazioni da compiere (lettura e scrittura di dati, operazioni aritmetiche e logiche, confronti, ...).

Le seconde, dette anche *pseudoistruzioni* o *direttive* o *istruzioni dichiarative*, non fanno avanzare il flusso logico del programma ma sono comandi diretti al traduttore, per esempio per definire il tipo di dati e la loro allocazione in memoria.

Le pseudoistruzioni DIM e DEF*tipo* sono state spiegate nei paragrafi 4.3.2 e 4.3.3 che trattano variabili e tabelle.

5.1 Commento

La pseudoistruzione

REM *commento*

oppure

' *commento*

dove *commento* indica un testo qualunque, è usata di solito per spiegare il significato di una parte di programma.

Se all'interno di una linea Basic compare la direttiva REM o il carattere apice, tutto quello che segue fino al termine della linea viene considerato commento.
Direttive siffatte possono essere inserite in qualunque punto del programma per migliorarne la leggibilità ma non vengono prese in considerazione durante la traduzione e l'esecuzione.

Esempi:

```
REM Programma per il calcolo della media aritmetica
INPUT n:  REM legge il numero n dei dati
a = (b * h) / 2 ' area del triangolo
```

5.2 Assegnazione

Un'istruzione dal formato

variabile = *espressione*

assegna alla variabile scritta a sinistra del segno di uguale il valore dell'espressione che compare sulla destra.

Il segno = ha quindi il significato di trasferimento di valore, non di uguaglianza.

Esempi:

```
a = d
a = (b * h) / 2
nome$ = "Topolino"
f$ = LEFT$(n$, 1) + RIGHT$(m$, 1)
```

Quando si debbano scambiare tra loro i contenuti di due variabili x e y è dunque sbagliato scrivere una coppia di istruzioni come
`x = y:  y = x`
poiché la prima assegnazione attribuisce a x lo stesso valore di y facendo perdere il valore di x originario e la seconda non fa che ribadire in y il valore che già contiene.
Per effettuare lo scambio bisogna far ricorso ad una terza variabile, da usare come appoggio temporaneo, oppure utilizzare l'istruzione

SWAP *var1*,*var2*

dove *var1* e *var2* sono le due variabili da scambiare.

Per scambiare i contenuti di x e y si farà dunque
`t = x:  x = y:  y = t`
oppure
`SWAP x,y`

5.3 Cancellazione dello schermo

L'istruzione

CLS

cancella il contenuto dello schermo e riporta il cursore in alto a sinistra.

5.4 Lettura di dati da tastiera

L'istruzione

INPUT ["*messaggio*"{;|,}] *v1* [,*v2*] ...

legge dei dati da tastiera e li memorizza ordinatamente nelle variabili di nome *v1*, *v2*, ... che possono essere variabili semplici o con indice oppure elementi di struttura definita dall'utente.

L'esecuzione della frase INPUT provoca la comparsa su video di un punto interrogativo e la sospensione del programma in attesa di dati. L'utente deve introdurre da tastiera i dati richiesti, separati da virgole se più di uno, e concludere battendo il tasto *Enter*. Fino a che *Enter* non viene battuto, i dati già scritti possono essere modificati.
Se i dati immessi non sono del tipo previsto nella lista di variabili o sono in numero diverso o non separati da virgole, compare la segnalazione d'errore `Redo from start` e l'utente deve immettere nuovamente tutti i dati.

L'argomento *messaggio* è un eventuale testo esplicativo che verrà emesso sullo schermo per richiedere i dati.

L'uso della virgola al posto del punto e virgola elimina l'emissione del punto interrogativo dopo il *messaggio*.

Esempi:
a lato di ogni istruzione è riportata la dicitura che compare su video per richiedere il dato.

```
INPUT N                                      ?
INPUT "Numero dei punti"; n                  Numero dei punti?
INPUT "Numero dei punti:", n                 Numero dei punti:
```

La funzione

INKEY$

(senza parametri) legge un singolo carattere da tastiera. Se non è stato battuto nessun tasto, la funzione restituisce la stringa nulla. Di conseguenza, essa viene comunemente usata in un ciclo di attesa che continua ad interrogare la tastiera fino a che viene premuto un tasto. Il programma presentato nel paragrafo 5.11 contiene un tipico esempio di utilizzo di INKEY$.

Si noti che, a differenza di quanto avviene con INPUT, il carattere letto con INKEY$ non viene emesso sullo schermo e non deve essere seguito da *Enter*.
Se dalla tastiera si immette più di un carattere (*Enter* è considerato alla stregua di qualsiasi altro), quelli eccedenti vengono assunti come dati per le eventuali INKEY$ successive.

5.5 Lettura di dati costanti con READ e DATA

In alcuni casi i programmi utilizzano molti dati che non variano da un'esecuzione all'altra. Un modo per fornire questi dati una volta per tutte evitando di introdurli da tastiera ad ogni esecuzione è quello di usare le istruzioni READ e DATA.

L'istruzione DATA ha la sintassi

DATA *d1* [,*d2*] ...

ove *d1*, *d2*, ... sono costanti di tipo numerico o stringa.
Le stringhe devono essere obbligatoriamente racchiuse tra doppi apici solo se tra i caratteri che le compongono compaiono virgole, due punti o spazi.

L'istruzione

READ *v1* [,*v2*] ...

assegna alle variabili *v1*, *v2*, ... i valori che preleva dalle frasi DATA, con corrispondenza uno a uno. Le variabili possono essere semplici o con indice oppure elementi di struttura definita dall'utente.

Un'unica frase READ può prelevare dati da più frasi DATA consecutive; parecchie READ possono usare la stessa DATA.
Se nella READ compaiono più variabili di quanti siano i valori presenti nelle DATA, viene segnalato l'errore di mancanza di dati `Out of DATA`.
Viceversa se la lista di variabili è composta da un numero di elementi inferiore al numero di dati presenti in DATA, la successiva READ inizierà la lettura dal primo dato non ancora letto. Se non esiste una READ successiva, i dati eccedenti sono ignorati.

Per rileggere valori specificati in alcune istruzioni DATA è disponibile l'istruzione

RESTORE [*etic*]

che fa sì che la prossima READ inizi a prelevare i dati dalla istruzione DATA con etichetta *etic*. Se *etic* è omessa, i dati vengono riletti dalla prima DATA.

Esempio:
l'esecuzione della seguente porzione di programma:

```
' Uso delle istruzioni READ e DATA
    DATA Gennaio,31
```

```
10    DATA Febbraio,28
      DATA Marzo,31,Aprile,30
      READ a1$, n1, a2$, n2, a3$, n3
      RESTORE 10
      READ b1$, l1, b2$, l2, b3$, l3
      ... ...
```

conduce ai seguenti valori di variabili:

a1$ = "Gennaio"	n1 = 30
a2$ = "Febbraio"	n2 = 28
a3$ = "Marzo"	n3 = 31
b1$ = "Febbraio"	l1 = 28
b2$ = "Marzo"	l2 = 31
b3$ = "Aprile"	l3 = 30

5.6 Scrittura di dati su video o su stampante

Le istruzioni

PRINT [*lista*] [{;|,}]

LPRINT [*lista*] [{;|,}]

scrivono rispettivamente su video e su stampante i dati specificati da *lista*.

L'argomento *lista* è costituito da una o più espressioni di cui si vuol stampare il valore; le espressioni possono essere di tipo numerico o stringa e sono separate l'una dall'altra dal carattere punto e virgola o virgola.

La posizione dei dati visualizzati viene determinata dai segni di punteggiatura usati per separare i dati in *lista*.
Un dato preceduto da un punto e virgola viene visualizzato subito dopo il dato precedente.

Un dato preceduto da una virgola viene stampato all'inizio della zona di scrittura successiva, immaginando la linea suddivisa in zone di 14 posizioni ciascuna.

Se l'istruzione termina con una virgola o con un punto e virgola, la **PRINT** successiva inizierà a visualizzare i dati sulla stessa linea dei precedenti, dopo aver spaziato di una quantità dipendente dal segno di punteggiatura finale della frase.

Una **PRINT** senza argomenti stampa una linea bianca ovvero effettua un "a capo".

Esempio:
l'esecuzione delle due istruzioni

```
INPUT k
PRINT "K="; k, "2*K="; 2 * k
```

dà luogo a:

```
? -3
K=-3          2*K=-6
```

5.7 Scrittura con formato

Le istruzioni

PRINT USING *stringaformato* ; *lista* [{;|,}]

LPRINT USING *stringaformato* ; *lista* [{;|,}]

permettono di assegnare un formato ai risultati da stampare rispettivamente su video e su stampante.

La *stringaformato* è una costante o variabile stringa che determina la forma e la disposizione dei dati.

La *lista* è composta dalle espressioni da stampare, separate indifferentemente da virgola o da punto e virgola.

Se l'istruzione termina con una virgola o con un punto e virgola, la

successiva istruzione di scrittura inizierà a stampare sulla stessa linea se c'è spazio a sufficienza.

I più comuni caratteri di formato per stampare valori numerici sono:

#	rappresenta la posizione di ciascuna cifra; se il numero ha meno cifre di quelle specificate, viene allineato sulla destra del campo facendolo precedere da spazi;
.	rappresenta il punto decimale, che può essere inserito in qualunque posizione del campo; se il dato ha più cifre di quelle previste dal formato, esso viene arrotondato;
+	se viene posto all'inizio della stringa di formato, permette di stampare il segno del numero;
^^^^	indicano il formato esponenziale; possono essere inseriti dopo i caratteri "#" e forniscono lo spazio per l'esponente scritto nella forma E±xx;
spazio	permette di spaziare i valori stampati.

Esempi:

```
PRINT USING "+###.## "; a, b, c
```

Se a = 21.314, b = 132.216, c = −3.12, l'istruzione provoca la scrittura di

```
 +21.31 +132.22   -3.12
```

```
PRINT USING "A=##   B=###.##^^^^"; a, b
```

Se a = 15 e b = 1.376, si ottiene la scrittura

```
A=15   B= 13.76E-01
```

Sono disponibili due funzioni di libreria che possono essere utilizzate nelle frasi di scrittura PRINT e PRINT USING per regolare la stampa. Esse sono:

SPC(x)	sposta il cursore di x posizioni;
TAB(x)	sposta il cursore nella posizione x della riga di stampa.

Entrambe le funzioni possono essere inserite nelle frasi di scrittura come comuni espressioni.

Esempi:

```
PRINT "ALFA"; SPC(20); "BETA"
```



```
PRINT "ALFA"; TAB(20); "BETA"
```

Nel primo caso le due stringhe vengono stampate separate l'una dall'altra da 20 spazi; nel secondo caso la seconda stringa viene stampata a partire dalla 20-esima posizione della riga.

Inoltre, se si desidera che la stampa su video inizi in una posizione di riga e colonna prefissata, si può usare prima della PRINT l'istruzione

LOCATE *riga,colonna*

ove $1 \leq riga \leq 24$ e $1 \leq colonna \leq 80$.

Esempio:

```
LOCATE 5,1: PRINT "VALORI INIZIALI"
```

5.8 Termine dell'esecuzione di un programma

L'istruzione

END

può essere posta in qualsiasi punto del programma per terminarne l'esecuzione.

Può essere tralasciata quando è l'ultima del programma.

Esempio riassuntivo:
questo programma calcola le radici di una equazione di secondo grado nell'ipotesi che siano reali.

```
' Radici dell'equazione a*x^2+b*x+c
     INPUT "coefficienti A,B,C ", a, b, c
     r = SQR(b * b - 4 * a * c)
     x1 = (-b + r) / (2 * a): x2 = (-b - r) / (2 * a)
     PRINT
     PRINT "A="; a, "B="; b, "C="; c
     PRINT "X1="; x1, "X2="; x2
     END
```

L'esecuzione porta alla stampa dei seguenti dati e risultati:

```
coefficienti A,B,C 2,5,3
A=2            B=5            C=3
X1=-1          X2=-1.5
```

5.9 Sospensione di un programma

L'istruzione

STOP

sospende l'esecuzione del programma. In ambiente QuickBASIC esso può essere ripreso dal punto interrotto mediante un opportuno comando (v. paragrafo 6.9).

5.10 Istruzioni di controllo

Le linee di un programma vengono normalmente eseguite nell'ordine in cui si presentano scritte. Per alterare l'ordine di esecuzione si usano le cosiddette *istruzioni di controllo*, che fanno proseguire il programma da punti diversi a seconda del verificarsi di certe condizioni.

5.10.1 Salto incondizionato

L'istruzione

GOTO *etic*

fa proseguire l'esecuzione del programma dalla linea con etichetta *etic*.

5.10.2 Salto calcolato

Il cosiddetto *salto calcolato* è effettuato dall'istruzione

ON *esp* GOTO *listaetic*

dove *esp* è un'espressione numerica intera di valore compreso tra 1 e 255 e *listaetic* è un elenco di etichette di linea separate da virgole.

A seconda che *esp* valga 1, 2, ... l'esecuzione del programma prosegue dalla linea la cui etichetta compare come prima, seconda, ... in *listaetic*.
Se *esp* vale zero o è maggiore del numero di elementi in *listaetic*, l'esecuzione prosegue dalla istruzione successiva alla ON...GOTO.
Se *esp* assume un valore negativo o maggiore di 255, viene segnalato l'errore `Illegal function call`.

5.10.3 Salto condizionato

L'istruzione di salto condizionato ha due possibili costrutti.

Prima forma (su una sola linea):

IF *espl* THEN *istr-t* [ELSE *istr-e*]

ove *espl* è un'espressione logica e sia *istr-t* che *istr-e* indicano una o più istruzioni separate fra loro dal carattere due punti.

Se la condizione espressa da *espl* è vera, vengono eseguite le istruzioni *istr-t*, altrimenti questa parte viene ignorata e vengono eseguite, se presenti, le istruzioni *istr-e*. L'esecuzione procede poi dall'istruzione logicamente successiva.

Esempi:

`IF a = 2 THEN GOTO 30`
Se il valore di a è 2, si prosegue dalla linea 30, altrimenti si passa alla linea successiva.

`a = 1:  IF k THEN a = a + 2`
Se k è vero, il valore di a viene incrementato di 2, altrimenti rimane eguale a 1.

`IF x < y THEN z = z + x ELSE z = z + y`
La variabile z viene incrementata del valore minore tra x e y.

Nell'esempio del paragrafo 5.8 per il calcolo delle radici di un'equazione di secondo grado si facciano le seguenti modifiche:
- si ponga l'etichetta 20 alla frase `INPUT`,
- dopo tale linea se ne inserisca una nuova con `IF a = 0 THEN END`,

- si sostituisca l'istruzione `END` con `GOTO 20`.
Il programma così modificato esegue il calcolo per più equazioni fino a che viene fornito un valore nullo per il coefficiente del termine di grado massimo.

Seconda forma (a blocchi):

IF *espl1* THEN
  [*frasi1*]
 [ELSEIF *espl2* THEN
  [*frasi2*]]
 ... ...
 [ELSE
  [*frasin*]]
END IF

ove *espl1*, *espl2*, ... sono espressioni logiche e *frasi1*, *frasi2*, ..., *frasin* individuano blocchi di istruzioni Basic su una o più linee.

Se la condizione espressa da *espl1* è vera vengono eseguite le istruzioni del blocco *frasi1*. Se *espl1* è falsa si esaminano in successione *espl2*, *espl3*, ... Alla prima condizione vera, vengono eseguite le frasi del blocco THEN ad essa associato. Se sono tutte false, vengono eseguite le istruzioni del blocco *frasin*. In ogni caso il programma continua con la frase successiva a END IF.

Esempio:
il programma seguente legge singoli caratteri da tastiera e per ognuno determina il tipo (cifra, lettera maiuscola o minuscola, altro), terminando quando viene immessa la stringa nulla.

```
' Determinazione del tipo di un carattere
10    INPUT c$
      IF c$ = "" THEN END
      IF LEN(c$) <> 1 THEN GOTO 10
      IF c$ >= "0" AND c$ <= "9" THEN
        PRINT "numero"
        ELSEIF c$ >= "A" AND c$ <= "Z" THEN
          PRINT "lettera maiuscola"
```



```
        ELSEIF c$ >= "a" AND c$ <= "z" THEN
          PRINT "lettera minuscola"
      ELSE
        PRINT "altro carattere"
    END IF
    GOTO 10
```

5.10.4 Ciclo

Esistono quattro strutture di controllo che permettono di ripetere più volte un insieme di istruzioni:

a) FOR...NEXT

b) WHILE...WEND

c) DO {WHILE | UNTIL}...LOOP

d) DO...LOOP {WHILE | UNTIL}

La struttura a) ha la seguente sintassi:

FOR *cont* = *inizio* TO *fine* [STEP *passo*]
  [*frasi*]
NEXT [*cont*]

Il costrutto permette di ripetere un numero specificato di volte l'insieme di istruzioni comprese tra FOR e NEXT.
Il numero di ripetizioni è determinato dal valore delle tre espressioni numeriche *inizio*, *fine* e *passo*.
Se *passo* non è specificato, viene assunto uguale a 1. Se è specificato, deve essere positivo nel caso $inizio < fine$, negativo nel caso contrario.

L'effetto del costrutto FOR...NEXT è il seguente. Alla variabile *cont*, detta *contatore del ciclo*, viene assegnato valore iniziale *inizio* e sono eseguite tutte le istruzioni che seguono fino a NEXT. Il contatore viene poi incrementato di *passo* e confrontato con *fine*. Se $cont \leq fine$ ($cont \geq fine$ in caso di passo negativo), viene ripetuto il ciclo a partire dall'istruzione che segue la FOR; altrimenti l'esecuzione prosegue dalla istruzione successiva a NEXT.

Il contatore può essere usato all'interno del ciclo ma non modificato.

Si noti che nel caso in cui la differenza *fine−inizio* ha segno discorde con quello di *passo* il ciclo non viene eseguito neppure una volta.

Esempio 1:
il seguente programma calcola e stampa la somma e il prodotto dei primi n interi positivi. Le due linee comprese tra le istruzioni **FOR** e **NEXT** costituiscono il corpo del ciclo, che viene ripetuto n volte per i= 1,2,...,n.

```
' Somma e prodotto dei primi n interi positivi
    DEFINT I, N, S
    INPUT n
    s = 0: p = 1
    FOR i = 1 TO n
      s = s + i
      p = p * i
    NEXT i
    PRINT "somma dei primi "; n; " interi ="; s
    PRINT "prodotto dei primi "; n; " interi ="; p
    END
```

Più cicli **FOR...NEXT** possono essere *annidati* l'uno nell'altro, cioè un ciclo può essere incluso nel corpo di un altro. Le variabili usate come contatori dei cicli annidati devono avere nomi differenti. Le **NEXT** di chiusura devono comparire nel programma in ordine inverso a quello in cui appaiono le corrispondenti **FOR**.

Una **NEXT** in cui la variabile *cont* venga omessa è considerata come istruzione di chiusura dell'ultimo ciclo **FOR** aperto.

Esempio 2:
la seguente porzione di programma calcola e stampa la matrice c prodotto delle due matrici a e b.
Il ciclo più interno, controllato dalla variabile k, esegue il prodotto scalare della riga i per la colonna j e viene eseguito per tutte le possibili coppie di valori i e j.


```
' Prodotto di matrici
    DIM a(2, 3), b(3, 4), c(2, 4)
'
    . . .   . . .
    FOR i = 0 TO 2
      FOR j = 0 TO 4
        c(i, j) = 0
        FOR k = 0 TO 3
          c(i, j) = c(i, j) + a(i, k) * b(k, j)
        NEXT k
      NEXT j
    NEXT i
    FOR i = 0 TO 2
      FOR j = 0 TO 4
        PRINT c(i, j);
      NEXT j
      PRINT
    NEXT i
```

Il costrutto **WHILE...END** ha la seguente sintassi:

> **WHILE** *cond*
> *frasi*
> **WEND**

Se la condizione *cond* è vera (cioè è diversa da zero), sono eseguite le istruzioni successive alla **WHILE** fino alla **WEND**. Si torna poi a controllare *cond* e, se è ancora vera, si ripete il ciclo. Quando *cond* è falsa, cioè uguale a zero, il programma prosegue con l'istruzione successiva a **WEND**.

Più cicli **WHILE...WEND** possono essere annidati l'uno nell'altro, cioè un ciclo può essere incluso nel corpo di un altro. Le **WEND** di chiusura devono comparire nel programma in ordine inverso a quello in cui appaiono le corrispondenti **WHILE**.

Esempio 3:
l'esempio 1 può essere scritto anche mediante **WHILE...WEND**; in questo caso l'uso del costrutto b) è equivalente a quello del costrutto

a), salvo che sono a carico dell'utente l'inizializzazione e l'incremento del contatore.

```
' Somma e prodotto dei primi n interi positivi
     DEFINT I, N, S
     INPUT n
     i = 1: s = 0: p = 1
     WHILE i <= n
       s = s + i
       p = p * i
       i = i + 1
     WEND
     PRINT "somma dei primi "; n; " interi ="; s
     PRINT "prodotto dei primi "; n; " interi ="; p
     END
```

Il costrutto b) è particolarmente utile per ripetere un ciclo di istruzioni un numero imprecisato di volte, come nell'esempio che segue.

Esempio 4:
il seguente programma calcola il massimo comun divisore con il metodo delle sottrazioni successive.

```
' MCD con sottrazioni successive
     INPUT n, m
     WHILE n <> m
       IF n > m THEN n = n - m ELSE m = m - n
     WEND
     PRINT "mcd="; n
     END
```

I costrutti c) e d) hanno rispettivamente la seguente sintassi:

DO [{WHILE | UNTIL} *espl*]
[*frasi*]
LOOP

e

DO
[*frasi*]
LOOP [{WHILE | UNTIL} *espl*]

Nella prima forma e con l'uso di WHILE (in alternativa UNTIL), se l'espressione logica *espl* è vera (falsa) si eseguono le frasi del ciclo fino a LOOP. Si ritorna poi ad esaminare *espl* e, se ancora vera (falsa), si ripete il ciclo. Quando *espl* diventa falsa (vera) si passa ad eseguire l'istruzione successiva a LOOP.

La seconda forma è analoga alla prima salvo il fatto che il controllo su *espl* viene eseguito alla fine del ciclo anziché all'inizio, quindi il ciclo è eseguito sempre almeno una volta.

Esempio 5:
il seguente programma calcola il massimo comun divisore con l'algoritmo euclideo:

```
' MCD con l'algoritmo di Euclide
     INPUT n, m
     PRINT "mcd("; n; ","; m; ")=";
     DO
       r = n MOD m
       n = m: m = r
     LOOP WHILE r <> 0    '(oppure LOOP UNTIL r = 0)
     PRINT n
     END
```

o con le sottrazioni successive:

```
' MCD con le sottrazioni successive
     INPUT n, m
     PRINT "mcd("; n; ","; m; ")=";
     DO WHILE n <> m    '(oppure DO UNTIL n = m)
       IF n > m THEN n = n - m ELSE m = m - n
     LOOP
     PRINT n
     END
```

Se da un punto qualunque di un programma si trasferisce il controllo ad una linea facente parte del corpo di un ciclo senza aver eseguito la frase iniziale **FOR**, **WHILE** o **DO**, possono capitare errori difficili da diagnosticare e correggere.

Nei costrutti **DO...LOOP** e **FOR...NEXT** per interrompere un ciclo indipendentemente dalla condizione che lo controlla si possono usare rispettivamente le istruzioni

EXIT DO

e

EXIT FOR

I programmi seguenti forniscono due esempi d'uso di **EXIT**.

Esempio 6:

```
' MCD con le sottrazioni successive
    INPUT n, m
    DO
      IF n = m THEN EXIT DO
      IF n > m THEN n = n - m ELSE m = m - n
    LOOP
    PRINT "mcd="; n
    END
```

Esempio 7:

```
' Legge un numero imprecisato (<= 1000) di dati e li
' somma via via fino a quando il dato letto e' = 0
    s = 0
    FOR i = 1 TO 1000
      INPUT n
      IF n = 0 THEN EXIT FOR
      s = s + n
    NEXT i
    PRINT s
    END
```


5.11 Salto ad un sottoprogramma

L'istruzione

GOSUB *etic*

trasferisce il controllo alla linea di programma con etichetta *etic*. Da qui l'esecuzione continua fino alla comparsa di

RETURN

che la fa riprendere dall'istruzione successiva alla **GOSUB**.

Questo metodo permette di usare lo stesso insieme di istruzioni (*sottoprogramma*) in punti diversi di un programma.

Un sottoprogramma può includere più di una frase **RETURN** se sono previste diverse vie di ritorno al programma chiamante.
Esso può essere collocato in un qualunque punto del programma ma il controllo vi deve essere trasferito solo tramite l'istruzione **GOSUB**. Perciò per evitare di eseguire il sottoprogramma senza aver eseguito la chiamata, la sua prima istruzione deve essere preceduta da una **GOTO** o da una **END**.

Un sottoprogramma ne può richiamare un altro ma non può richiamare sé stesso, neppure indirettamente.

Esempio:
questo programma realizza il gioco a due giocatori "indovina la parola". Un giocatore, di nascosto dal compagno, introduce la "parola segreta", che scompare immediatamente dal video. Il secondo giocatore deve cercare di indovinarla, mediante tentativi che riducono via via il campo delle possibilità.
Il sottoprogramma composto dalla linea 100 e dalla successiva scrive sulla riga corrente del video un certo carattere per 20 volte e poi effettua un "a capo".
Il carattere stampato è il punto quando il programma fornisce al giocatore il nuovo intervallo di scelta, il punto interrogativo quando il tentativo del giocatore è fuori da tale intervallo, il punto esclamativo quando la parola segreta viene indovinata.

Si noti anche il ciclo di interrogazione della tastiera mediante la funzione INKEY$.

```
' Indovina la parola
10    nt = 0: ini$ = "a": fin$ = "zuzzurellone"
      INPUT "PAROLA SEGRETA"; par$: CLS
30    nt = nt + 1
40    PRINT "TENTATIVO "; nt; " - TRA "; ini$; " E "; fin$;
      c$ = ".": GOSUB 100: INPUT ten$
      IF ten$ = par$ THEN
        PRINT "HAI INDOVINATO"; : c$ = "!"
        GOSUB 100: PRINT "VUOI GIOCARE ANCORA?";
        DO
          a$ = INKEY$
        LOOP WHILE a$ = ""
        IF UCASE$(a$) = "S" THEN GOTO 10 ELSE END
      END IF
      IF ten$ <= ini$ OR ten$ >= fin$ THEN
        c$ = "?": GOSUB 100: GOTO 40
      END IF
      IF ten$ < par$ THEN ini$ = ten$ ELSE fin$ = ten$
      GOTO 30
100   FOR I = 1 TO 20: PRINT c$;: NEXT I
      PRINT: RETURN
```

Di seguito viene riportata la lista di una esecuzione:

```
PAROLA SEGRETA? giallo
TENTATIVO  1  - TRA a E zuzzurellone....................
?  verde
TENTATIVO  2  - TRA a E verde....................
?  blu
TENTATIVO  3  - TRA blu E verde....................
?  viola
????????????????????
TENTATIVO  3  - TRA blu E verde....................
?  giallo
```



```
HAI INDOVINATO!!!!!!!!!!!!!!!!!!!!
VUOI GIOCARE ANCORA?
```

5.12 Funzioni definite dall'utente

L'utente può definire funzioni da utilizzare nel corso del programma in modo analogo alle funzioni di libreria viste in 4.4.5. La definizione, che deve precedere il richiamo, viene fatta mediante l'istruzione DEF FN, che può estendersi su una o più linee, secondo le due sintassi seguenti:

a) DEF FN*nome* [(*listapar*)] = *espressione*

b) DEF FN*nome* [(*listapar*)]

   ... ...

   FN*nome* = *espressione*

   ... ...

   END DEF

nome è un identificatore che segue le regole dei nomi di variabili anche per quanto riguarda la dichiarazione di tipo;

listapar è l'elenco degli argomenti formali avente il seguente formato:

> *var* [AS *tipo*] [,*var* [AS *tipo*]] ...
>
> *var* è il nome di una qualunque variabile semplice;
>
> *tipo*, se presente, ne individua il tipo (INTEGER, LONG, ...); in alternativa il tipo della variabile può essere individuato dall'ultimo carattere del nome o da una precedente dichiarazione DEF*tipo*;

espressione è una espressione che, oltre ai parametri formali, può contenere altre variabili definite nel programma.

Nella seconda forma sintattica la frase FN*nome* = *espressione* deve comparire almeno una volta nel corpo della funzione per attribuire un valore al risultato.

Per richiamare una funzione basta che in una frase di assegnazione compaia, tra gli operandi dell'espressione, FN*nome* seguito dalla lista dei parametri attuali posta fra parentesi.
Nel calcolo gli argomenti formali che compaiono nell'espressione che definisce la funzione vengono sostituiti da quelli attuali; per le eventuali altre variabili viene usato il valore corrente ed una loro variazione si ripercuote nel loro uso in tutto il programma.
Quando viene richiamata una funzione a più linee, vengono eseguite le istruzioni che ne compongono il corpo fino alla END DEF. Uscite alternative possono essere effettuate tramite l'istruzione

EXIT DEF

Esempio 1:
il primo esempio mostra l'uso di una funzione a una frase, ovvero definita secondo la prima sintassi.

```
' Funzione a una frase
    DEF FNA (X) = SQR((SIN(X) + COS(X)) / 2)
    PIG = 4 * ATN(1)
    PRINT FNA(PIG / 3)
```

Una volta eseguito dà luogo al seguente risultato:

```
 .8264458
```

Esempio 2:
il secondo esempio utilizza una funzione a più frasi, cioè definita mediante la seconda sintassi, e prevede l'uscita dopo l'esecuzione completa del corpo della funzione.

```
' Funzione a piu' frasi
     DEFINT A-Z
     DEF fnmcd (n, m)
       WHILE n <> m
         IF n > m THEN n = n - m ELSE m = m - n
       WEND
       fnmcd = n
     END DEF
```



```
     INPUT i, j
     PRINT fnmcd(i, j)
     END
```

Esempio 3:
il terzo esempio tabula i valori di

$$y(x) = \begin{cases} \sin(1/x) & \text{se } x \neq 0 \\ 0 & \text{se } x = 0 \end{cases}$$

in un intervallo assegnato. La funzione ha due possibili punti d'uscita.

```
' Funzione a piu' frasi
     DEF fny (x)
       IF x = 0 THEN fny = 0: EXIT DEF
       fny = SIN(1 / x)
     END DEF
     INPUT z1, z2
     FOR z = z1 TO z2 STEP 1
       PRINT z, fny(z)
     NEXT z
     END
```

5.13 Tipi di files: sequenziali e diretti

Quando un programma richiede la lettura di una grande quantità di dati, per evitare di doverli introdurre da tastiera ad ogni esecuzione è opportuno memorizzarli su disco in un file che il programma può leggere.
Ci sono diversi tipi di organizzazione di un file: si accenna qui solo ai files di tipo sequenziale e a quelli di tipo diretto.

Un *file sequenziale* può essere generato o da un programma o mediante un editore di testi.
I dati sono memorizzati nel file nell'ordine in cui sono scritti e devono essere letti nello stesso ordine.

Ciascuna riga scritta costituisce un *record*. I records di un file sequenziale possono avere lunghezze differenti.
Per accedere all'i-esimo record è necessario aver letto tutti i dati degli $i-1$ records precedenti.

I *files ad accesso diretto* (o *casuale*) possono essere generati solo da un programma e hanno records tutti della stessa lunghezza.
Si può accedere all'i-esimo record direttamente, senza aver letto i precedenti.

Per lavorare su un file è necessario prima di tutto *aprirlo*, cioè definirne il nome, il tipo di accesso, la lunghezza dei records, per poter poi accedere ai dati in esso contenuti.
Alla fine dell'utilizzo il file deve essere *chiuso*, cioè devono essere eliminate dalla memoria le informazioni predisposte all'apertura.

5.13.1 Apertura e chiusura di un file

L'istruzione

OPEN *file* [FOR *modo*] AS #*n* [LEN = *lung*]

compie l'operazione di *apertura* di un file, indispensabile per utilizzarlo da programma.

Gli argomenti dell'istruzione sono i seguenti:

file è una costante o variabile di tipo stringa che fornisce il pathname, completo o ridotto, del file;

modo è una parola che determina la modalità d'uso del file. Per un file sequenziale può essere:

INPUT — se il file deve essere letto; il file deve esistere e la lettura avverrà dall'inizio;

OUTPUT — se il file deve essere scritto; se il file non esiste, viene creato; se esiste, il contenuto precedente è perso;

APPEND — se si devono aggiungere record alla fine, in modo da non perdere il contenuto precedente; il file può essere solo scritto e se non esiste viene creato.

Per un file ad accesso diretto, *modo* è sempre RANDOM e l'opzione si può tralasciare;

n è un numero intero compreso tra 1 e 15, che viene associato al file e si usa poi nelle operazioni di lettura e scrittura;

lung vale solo per i files ad accesso diretto e indica la lunghezza del record in bytes.

Esempi:

```
OPEN NOME$ FOR INPUT AS #1
OPEN "RISULTATI.DAT" FOR OUTPUT AS #5
OPEN "DATI" AS #1 LEN = 10 o equivalentemente
OPEN "DATI" FOR RANDOM AS #1 LEN = 10
```

L'istruzione

CLOSE [[#] *n*]

compie l'operazione di *chiusura* del file associato al numero *n*. Se il numero di file non è specificato, chiude tutti i files attualmente aperti.

L'istruzione annulla la corrispondenza tra file e numero attribuito con OPEN, quindi l'accesso da programma ai dati contenuti nel file può avvenire solo dopo una nuova apertura.

Un file che sia stato precedentemente chiuso può essere riaperto assegnandogli lo stesso numero o uno diverso.
Per un file sequenziale si può modificare anche il modo di accesso. Per esempio, per generare un file sequenziale e successivamente rileggerlo nel corso dello stesso programma, prima il file deve essere aperto in modo OUTPUT, poi, terminata la scrittura, deve essere chiuso e successivamente riaperto in modo INPUT.

5.13.2 Lettura e scrittura di files sequenziali

Per leggere i dati contenuti in un file sequenziale si usa l'istruzione

INPUT #*n*, *lista*

n è il numero attribuito al file con l'istruzione OPEN in modo INPUT;

lista specifica, separati da virgola, i nomi delle variabili alle quali vanno assegnati i valori letti.

Ogni istruzione INPUT # esegue la lettura di un record, perciò le variabili della lista devono corrispondere in numero e tipo ai dati presenti nel record.

Come per la lettura da tastiera, se la lista contiene più di una variabile i dati corrispondenti devono essere stati memorizzati nel record separati da virgole. Come separatore si può usare anche lo spazio; in questo caso però ogni stringa deve essere racchiusa tra doppi apici.

Il tentativo di lettura di un file sequenziale al di là dell'ultimo record fa terminare l'esecuzione del programma con una segnalazione di errore. Questo può essere evitato controllando, prima di eseguire ogni lettura, se è stata raggiunta o no la fine del file. A tale scopo si usa la funzione

EOF(*n*)

(ove *n* è il numero del file) che assume valore −1 (vero) se si è raggiunta la fine del file, 0 (falso) altrimenti.

L'istruzione di scrittura su file sequenziale può avere una delle tre seguenti sintassi:

a) PRINT #*n*, *lista*

b) PRINT #*n*, USING *stringaformato*; *lista*

c) WRITE #*n*, *lista*

n è il numero associato al file con l'istruzione OPEN in modo OUTPUT o APPEND;

lista è una lista di espressioni i cui valori verranno scritti in un record del file. Nella lista le espressioni compaiono separate dai segni di punteggiatura già descritti per le istruzioni PRINT e PRINT USING;

anche *stringaformato* ha significato analogo a quello già visto.

Usando le istruzioni di scrittura di tipo a) e b) l'utente deve preoccuparsi di scrivere nel record anche i separatori tra i dati, in modo da consentirne una corretta lettura successiva.
L'istruzione WRITE invece inserisce automaticamente i separatori opportuni: i dati vengono separati da virgole e le stringhe vengono racchiuse tra doppi apici.

Esempio:
siano `a$="alfa"`, `b$="beta"`, e `c=4` tre dati da memorizzare in un record. Le tre frasi seguenti costituiscono tre modi validi affinché una successiva istruzione `INPUT #n, a$, b$, c` legga i valori corretti.

```
PRINT #n, a$; ","; b$; ","; c
```
scrive sul file n → `alfa,beta, 4`

```
PRINT #n, CHR$(34);a$;CHR$(34);CHR$(34);b$;CHR$(34);c
```
scrive sul file n → `"alfa""beta" 4`

```
WRITE #n, a$; b$; c
```
scrive sul file n → `"alfa","beta",4`

Si noti che il carattere doppio apice non può far parte di una costante stringa, perciò per scrivere nel record i delimitatori delle stringhe si usa la funzione `CHR$(34)`, essendo 34 il codice ASCII corrispondente al doppio apice.

Esempio riassuntivo:
il programma seguente mostra l'uso delle istruzioni di gestione dei files sequenziali. Legge da tastiera coppie di valori interi finché non sono entrambi nulli, li scrive sul file VAL.DAT quindi li rilegge dallo stesso file e li stampa incolonnati.

```
' Gestione di files ad accesso sequenziale
    DEFINT I
    OPEN "VAL.DAT" FOR OUTPUT AS #3
    INPUT i1, i2
    WHILE i1 <> 0 AND i2 <> 0
      WRITE #3, i1, i2
      INPUT i1, i2
    WEND
```

```
CLOSE #3
OPEN "VAL.DAT" FOR INPUT AS #1
WHILE NOT EOF(1)
  INPUT #1, i1, i2
  PRINT USING "#####"; i1; i2
WEND
CLOSE #1
END
```

5.13.3 Lettura e scrittura di files ad accesso diretto

Il modo più semplice per leggere e scrivere files ad accesso diretto è definire mediante l'istruzione TYPE (vedi paragrafo 4.3.4) un tipo di dato contenente tutte le variabili che costituiscono un record e, mediante DIM, dichiarare poi una o più variabili con la struttura definita da TYPE.

Per scrivere un record basta allora usare l'istruzione

PUT #*n*, *nrec*, *var*

dove

n è il numero associato al file con l'istruzione OPEN;
nrec è il numero del record su cui si vuole scrivere;
var è la variabile con struttura definita dall'utente e contenente i dati da scrivere.

In modo analogo, per leggere dal record *nrec* si userà l'istruzione

GET #*n*, *nrec*, *var*

I singoli campi del record sono accessibili come visto nella descrizione della frase TYPE.

Esempio:
il seguente programma scrive su un file dei records contenenti ciascuno un nome lungo al massimo 20 caratteri ed un punteggio espresso come

numero reale in semplice precisione; poi rilegge i dati e li trascrive sullo schermo.

```
' Gestione di files ad accesso diretto
    TYPE test
      nome AS STRING * 20
      punti AS SINGLE
    END TYPE
    DIM rec AS test
    OPEN "dati" FOR RANDOM AS #1 LEN = LEN(rec)
    rec.punti = 100: i = 1
' legge i dati da tastiera e li scrive sul file ad
' accesso casuale "dati"
    WHILE rec.punti <> 0
      INPUT "nome,punti"; rec.nome, rec.punti
      PUT #1, i, rec
      i = i + 1
    WEND
' legge da "dati" i record scritti e li stampa
' nrec e' il numero dei record del file
    nrec = LOF(1) / LEN(rec)
    FOR i = 1 TO nrec
      GET #1, i, rec
      LPRINT rec.nome, rec.punti
    NEXT i
    CLOSE #1
    END
```

Si noti che la funzione LEN(*var*) restituisce la lunghezza in bytes della variabile *var* e la funzione LOF(*n*) dà la lunghezza in bytes del file associato al numero *n*.

5.14 Istruzioni grafiche

Il linguaggio Basic fornisce un insieme di istruzioni e funzioni di grafica che consentono di visualizzare punti e linee sul video. Verranno qui descritte solo le istruzioni più utili, nei loro formati più comuni.

Quando si entra in ambiente QuickBASIC si è in *modo testo*, ovvero si possono visualizzare, mediante le istruzioni descritte in precedenza, solo caratteri nelle 24 righe e 80 colonne dello schermo.

Nel *modo grafico*, invece, lo schermo è visto come una griglia di punti che possono essere accesi o spenti singolarmente. Il numero di punti della griglia, in orizzontale ed in verticale, si chiama *risoluzione* dello schermo e dipende dalla particolare scheda grafica in dotazione del personal.

Presso il Laboratorio Didattico sono presenti schede Hercules e VGA, con risoluzione rispettivamente 720×348 e 640×480. La scheda VGA supporta anche i modi EGA (640×350) e CGA (320×200 o 640×200). I monitor sono tutti monocromatici (bianco/nero); pertanto nelle spiegazioni delle istruzioni non verranno prese in considerazione le opzioni relative ai colori.

Per poter usare le istruzioni di grafica occorre selezionare mediante una opportuna istruzione il modo grafico, altrimenti l'esecuzione di una di tali istruzioni conduce ad una segnalazione d'errore.

5.14.1 Attivazione e disattivazione della grafica

L'istruzione

SCREEN n (con $n > 0$)

attiva il modo grafico. Il valore di n dipende dalla scheda grafica in dotazione.

La tabella seguente riporta i valori di n validi per i PC del Laboratorio con le rispettive risoluzioni in modo grafico e testo.

n	*modo*	*grafico*	*testo*
3	Hercules	720×348	80×24
7	CGA	320×200	40×24
8	CGA	640×200	80×24
10	EGA	640×350	80×24
12	VGA	640×480	80×30

Nel modo grafico, oltre alle istruzioni di grafica, sono disponibili tutte le istruzioni e le funzioni disponibili nel modo testo per visualizzare sequenze di caratteri.

L'istruzione

SCREEN 0

attiva il modo testo, disattivando il modo grafico.

5.14.2 Coordinate fisiche

I grafici si ottengono "accendendo" sul video i punti opportuni di una griglia discreta dello schermo. Un punto, detto anche *pixel*, è individuato dalle coordinate (x, y) che ne forniscono la posizione sulla griglia. Tali coordinate sono dette *coordinate fisiche* e assumono valori interi positivi tali che

$$0 \leq x \leq nx - 1$$

$$0 \leq y \leq ny - 1$$

ove $nx \times ny$ è la risoluzione fornita dalla scheda grafica.

Il punto (0,0), origine degli assi, è situato nel vertice in alto a sinistra dello schermo, l'asse x è orientato verso destra, l'asse y verso il basso.

5.14.3 Coordinate logiche

All'utente riesce poco spontaneo ragionare nei termini restrittivi delle coordinate fisiche, soprattutto in campo matematico, dove la grafica viene usata in questioni del tipo: "vedere l'andamento della funzione $y = f(x)$ in un assegnato intervallo della x ...".

Il problema grafico di questo utente è generalmente posto in termini di coordinate reali, sia positive sia negative, con gli assi cartesiani orientati nel modo consueto, con i valori delle ascisse compresi tra due estremi arbitrari x_{min} e x_{max} e i valori delle ordinate tra y_{min} e y_{max}.

Questo sistema cartesiano di riferimento, proprio dell'utente e del particolare grafico, si chiama sistema di *coordinate logiche* e il rettangolo

individuato dai quattro valori estremi delle coordinate *finestra logica*. Per esempio, una finestra logica per il grafico di $y = \sin(x)$ può essere compresa tra 0 e 2π per le ascisse e tra -1.1 e 1.1 per le ordinate.

Per poter effettivamente tracciare sullo schermo il grafico idealmente situato nella finestra logica, l'utente può scegliere di effettuare personalmente la trasformazione da coordinate logiche a coordinate fisiche, inserendo opportune istruzioni nel programma; altrimenti può, più semplicemente, usare solo coordinate logiche e lasciare la trasformazione a carico del linguaggio, dopo aver dichiarato esplicitamente (con l'istruzione WINDOW - v. 5.14.6) gli estremi della finestra logica. Nel seguito saranno esaminati entrambi i casi.

5.14.4 Finestra fisica

Si chiama *finestra fisica* la porzione rettangolare di schermo utilizzata per visualizzare i grafici. Se non specificato altrimenti, la finestra fisica è l'intero schermo.

Si può alterare questa assegnazione standard mediante l'istruzione

VIEW (x_1,y_1) - (x_2,y_2)

dove (x_1, y_1) e (x_2, y_2) sono le coordinate fisiche di due vertici diagonalmente opposti del rettangolo costituente la finestra.

Esempio:
```
VIEW  (0,0) - (360,174)
VIEW  (0,0) - (320,240)
```
definiscono come finestra fisica grafica il quarto superiore sinistro dello schermo rispettivamente per una scheda Hercules e per una VGA.

Ad ogni esecuzione di una istruzione VIEW viene definita una finestra fisica che è la finestra corrente su cui si può operare. Per cambiare finestra è necessario eseguire un'altra VIEW.

Una istruzione VIEW senza argomenti riporta la finestra fisica a coincidere con tutto lo schermo.

Se nelle istruzioni grafiche successive ad una VIEW si usano coordi-

nate fisiche, i loro valori sono considerati relativi al vertice in alto a sinistra della finestra fisica definita (assunto dunque come nuovo punto origine (0,0)).

5.14.5 Finestra di testo

Oltre a definire una o più finestre fisiche è possibile definire una *finestra di testo*, cioè riservare un'area rettangolare su cui visualizzare solo caratteri. La definizione avviene mediante l'istruzione

VIEW PRINT n TO m

ove n e m sono i numeri della prima e dell'ultima riga di schermo che limitano la finestra.

Quando il testo visualizzato raggiunge l'ultima riga della finestra, viene fatto scorrere verso l'alto della finestra stessa.

Esempio:
```
VIEW PRINT 1 TO 10
```
riserva per il testo le prime 10 righe dello schermo.

5.14.6 Finestra logica

Come accennato nel paragrafo 5.14.3, l'utente può definire un suo sistema di riferimento cartesiano, in modo da poter usare come coordinate numeri reali sia positivi sia negativi (coordinate logiche), svincolandosi dalle coordinate fisiche dello schermo. Più precisamente, può definire una *finestra logica* fornendo i valori x_{min}, y_{min}, x_{max} e y_{max} previsti come valori minimi e massimi delle ascisse e delle ordinate.

La definizione avviene mediante l'istruzione

WINDOW (x_{min},y_{min}) - (x_{max},y_{max})

Più in generale, le coppie di punti da indicare nell'istruzione rappresentano le coordinate logiche di due vertici diagonalmente opposti del rettangolo costituente la finestra.

Dopo l'esecuzione di una WINDOW, le coordinate di tutti i punti

specificati nelle istruzioni di grafica (**LINE**, **PSET**, **PRESET**) sono interpretate dal Basic come coordinate logiche e convertite automaticamente in coordinate fisiche. La finestra logica viene "proiettata" così sulla finestra fisica corrente.

Esempi:

```
WINDOW  (-500,-100) - (400,300)
```
La finestra logica $-500 \leq x \leq 400 \quad -100 \leq y \leq 300$ viene proiettata sull'intero schermo.

```
VIEW  (0,0) - (360,174):  WINDOW  (-10,-10) - (10,10)
```
La finestra logica $-10 \leq x \leq 10 \quad -10 \leq y \leq 10$ viene proiettata nel quarto superiore sinistro dello schermo (scheda Hercules).

5.14.7 Cancellazione del contenuto di finestre

In modo grafico l'istruzione **CLS** permette di cancellare il contenuto di una finestra o dell'intero video. Ha il seguente formato:

CLS [n]

ove n può assumere i valori 0, 1, 2.

CLS	senza parametro, cancella il contenuto della finestra corrente, grafica o di testo. Se non è stata definita nessuna finestra, questa istruzione cancella tutto lo schermo.
CLS 0	cancella l'intero schermo.
CLS 1	cancella il contenuto della finestra fisica corrente.
CLS 2	cancella il contenuto della finestra di testo corrente.

5.14.8 Accensione e spegnimento di punti sullo schermo

Per "accendere" il punto di coordinate (x, y) si usa l'istruzione

PSET (x,y)

Per "spegnerlo"

PRESET (x,y)

5.14.9 Tracciamento di linee e rettangoli

L'istruzione

LINE [(x_1,y_1)] - (x_2,y_2) [, , [B][,*tratteggio*]]

traccia un segmento congiungente i punti (x_1, y_1) e (x_2, y_2). Se (x_1, y_1) viene omesso, l'estremo iniziale del segmento è l'ultimo punto riferito con **PSET** o **PRESET** o come estremo finale di una **LINE** precedente.

L'argomento *tratteggio* indica la modalità di tracciamento. Se manca, tutti i punti sulla linea sono illuminati. Se specificato, è una *maschera* di 16 bit, fornita in esadecimale, che specifica l'alternanza di punti accesi (bit=1) o mantenuti inalterati (bit=0) nella linea.

La specifica **B** permette di disegnare un rettangolo (con i lati paralleli ai bordi dello schermo) con un'unica istruzione anziché quattro; in questo caso (x_1, y_1) e (x_2, y_2) sono le coordinate di due vertici diagonalmente opposti.

Esempio 1:
le seguenti istruzioni tracciano gli assi della finestra logica con linea continua.

```
SCREEN 12
WINDOW (-10,-10) - (10,10)
LINE (-10,0) - (10,0)
LINE (0,-10) - (0,10)
```

Esempio 2:
queste altre disegnano un triangolo con i lati tratteggiati; infatti la maschera binaria corrispondente al valore esadecimale F0F0 (che va scritto preceduto dai caratteri &H) è 1111000011110000, quindi il tratteggio prevede l'alternanza di 4 punti accesi e 4 spenti.

```
SCREEN 12
WINDOW (-10,-10) - (10,10)
LINE (-5,0) - (0,3) ,,,&HFOFO
LINE - (2,-2) ,,,&HFOFO
LINE - (-5,0) ,,,&HFOFO
```

5.14.10 Grafici in coordinate fisiche

Come già detto, se $nx \times ny$ è la risoluzione della scheda grafica le coordinate fisiche assumono valori compresi tra 0 e $nx - 1$ per le ascisse e tra 0 e $ny - 1$ per le ordinate; il punto (0,0) corrisponde al vertice in alto a sinistra dello schermo, per cui l'asse y ha orientamento opposto rispetto alla consuetudine.

Per tracciare un grafico, l'utente che voglia operare in coordinate fisiche ma con gli assi orientati secondo la consuetudine deve quindi provvedere a trasformare le coordinate (x, y) dei suoi punti in coordinate fisiche (x_F, y_F).

Se i valori dell'utente sono compresi tra x_{min} e x_{max} in ascisse e tra y_{min} e y_{max} in ordinate e il rettangolo individuato da questi valori deve essere proiettato sull'intero schermo, le coordinate fisiche corrispondenti ad un punto (x, y) vanno calcolate mediante le formule:

$$x_F = (x - x_{min}) \frac{(nx - 1)}{(x_{max} - x_{min})}$$

$$y_F = (y_{max} - y) \frac{(ny - 1)}{(y_{max} - y_{min})}$$

Se invece si definisce mediante una istruzione

VIEW (x_a, y_a) - (x_b, y_b) (con $x_a < x_b$, $y_a < y_b$)

una finestra fisica più piccola dello schermo, il grafico può essere proiettato su di essa mediante le formule:

$$x_F = (x - x_{min}) \frac{(x_b - x_a)}{(x_{max} - x_{min})}$$

$$y_F = (y_{max} - y) \frac{(y_b - y_a)}{(y_{max} - y_{min})}$$

In tal caso le coordinate fisiche (x_F, y_F) sono relative al punto origine della finestra (v. 5.14.4).

Esempio:
il seguente programma traccia sull'intero schermo (scheda VGA) il grafico della funzione $y = \sin(x)$ tra 0 e 2π, incrementando via via di 0.1 il valore di x.

```
' Grafico di y=sin(x) (coordinate fisiche)
    SCREEN 12: CLS
    xmin = 0: xmax = 8 * ATN(1)
    ymin = -1.1: ymax = 1.1
    fx = 639 / (xmax - xmin): fy = 479 / (ymax - ymin)
    xf = xmin: y = SIN(xmin): yf = (ymax - y) * fy
    PSET (xf, yf)
    FOR x = xmin TO xmax STEP .1
      xf = (x - xmin) * fx
      y = SIN(x)
      yf = (ymax - y) * fy
      LINE -(xf, yf)
    NEXT
    END
```

Se si aggiungesse dopo la frase `SCREEN 12` la linea
`VIEW (100,50) - (320,240)`
per tracciare il grafico nella finestra fisica scelta i fattori di conversione dovrebbero essere così modificati:
`fx = 220 / (xmax - xmin): fy = 190 / (ymax - ymin)`

Osservazione

Poiché il numero di punti e la loro distanza reciproca sono diversi lungo i due assi, il sistema di riferimento non è monometrico, cioè le unità di misura sui due assi sono diverse.

5.14.11 Grafici in coordinate logiche

L'istruzione WINDOW permette all'utente di usare direttamente il suo sistema di coordinate senza bisogno di programmare le trasformazioni viste nel paragrafo precedente.

Esempio:
il programma del paragrafo 5.14.10 può semplicemente essere riscritto nel modo seguente:

```
' Grafico di y=sin(x) (coordinate logiche)
     SCREEN 12: CLS
     xmin = 0: xmax = 8 * ATN(1)
     ymin = -1.1: ymax = 1.1
     WINDOW (xmin, ymin)-(xmax, ymax)
     x = xmin: y = SIN(x)
     PSET (x, y)
     FOR x = xmin TO xmax STEP .1
       y = SIN(x)
       LINE -(x, y)
       NEXT
     END
```

Se poi si volesse proiettare il grafico solo su una parte dello schermo, basterebbe aggiungere la frase VIEW che definisce la finestra fisica desiderata.

Esempio:
`VIEW  (100,50) - (320,240)`

Osservazione

Anche in questo caso il sistema cartesiano di riferimento non è monometrico.

Se si vuole che il sistema diventi monometrico, è necessario usare un fattore di scala, che dipende dal particolare tipo di video.
Per esempio, i video dei personal del Laboratorio Didattico con scheda Hercules hanno una pagina grafica di 22 × 15 cm.

Di conseguenza, per ottenere lungo l'asse y la stessa unità di misura dell'asse x, se si usa come finestra fisica l'intero schermo bisogna scegliere la finestra logica in modo che risulti

$$y_{max} - y_{min} = \frac{15}{22}(x_{max} - x_{min})$$

Se si definisce una finestra fisica più piccola, al posto dei coefficienti 15 e 22 bisogna usare le misure dei lati verticali e orizzontali della finestra.

Esempio:
il seguente programma traccia il grafico di $y = \sin(x)$ tra 0 e 2π in un sistema di riferimento cartesiano monometrico.

```
' Grafico di y=sin(x) (coord. logiche, assi monometr.)
     SCREEN 3: CLS
     xmin = 0: xmax = 8 * ATN(1)
     f = 15 * (xmax - xmin) / 22
     ymax = f / 2: ymin = -ymax
     WINDOW (xmin, ymin)-(xmax, ymax)
     x = xmin
     y = SIN(x)
     PSET (x, y)
     FOR x = xmin TO xmax STEP .1
       y = SIN(x)
       LINE -(x, y)
     NEXT
     END
```

5.15 Procedure e moduli

Un programma può essere suddiviso, per semplificarne la stesura, in componenti logiche più piccole dette *procedure*. Le procedure sono particolarmente utili quando si debba ripetere più volte lo stesso insieme di operazioni. Invece di riscrivere le stesse istruzioni, è opportuno iso-

larle in una procedura che può essere richiamata più volte, anche con dati differenti, là dove è necessario nel corso del programma.

Le procedure sono particolarmente vantaggiose per i seguenti motivi:

- possono essere provate singolarmente;
- sono utilizzabili da più programmi e quindi aumentano la potenzialità del linguaggio, in quanto possono costituire librerie a disposizione degli utenti;
- il programma generato ha lunghezza minore.

Si definiscono due tipi di procedure: quelle di tipo **SUB** e quelle di tipo **FUNCTION**.
Per molti versi queste procedure hanno un comportamento simile, rispettivamente, ai sottoprogrammi e alle funzioni definiti nei paragrafi 5.11 e 5.12 ma permettono maggiore elasticità d'uso.

Se un programma è costituito da un programma principale e da una o più procedure, il testo può essere memorizzato tutto in un unico file oppure in più files separati detti *moduli*.

Un modulo può contenere il programma principale, oppure il programma principale con tutte o alcune procedure da esso richiamate, oppure una o più procedure.
All'interno di uno stesso modulo ogni etichetta deve essere unica. Le variabili di ciascuna procedura sono invece considerate *locali* alla procedura stessa (a meno di diverse indicazioni); pertanto due procedure dello stesso modulo possono usare lo stesso nome di variabile con significati differenti.

Nella configurazione minima un programma consiste di almeno un modulo, contenente il programma principale.

La memorizzazione delle procedure in files separati facilita il loro uso da parte di più programmi.

5.15.1 Definizione di una procedura

La sintassi delle procedure di tipo **SUB** è la seguente:

> SUB *nomesub* [(*listapar*)]
> ... ...
> END SUB

ove *nomesub* è un identificatore lungo fino a 40 caratteri alfanumerici, il primo dei quali alfabetico. Questo nome non può comparire come nome di altra **SUB** o **FUNCTION** nello stesso programma o nella libreria dell'utente.

La sintassi delle procedure di tipo **FUNCTION** è la seguente:

> FUNCTION *nomefun* [(*listapar*)]
> ... ...
> *nomefun* = *espressione*
> ... ...
> END FUNCTION

ove *nomefun* è un identificatore che segue le regole dei nomi di variabili anche per quanto riguarda la dichiarazione di tipo. Il tipo di *nomefun* determina il tipo del valore ritornato dalla funzione.
La frase *nomefun* = *espressione* deve comparire almeno una volta nel corpo della funzione per attribuire un valore al risultato.

In entrambi i casi *listapar* evidenzia il numero e il tipo di argomenti da trasmettere alla procedura ed ha la seguente sintassi:

> *var* [( )] [AS *tipo*] [,*var* [( )] [AS *tipo*]] ...

> *var* è il nome di una qualunque variabile; i nomi di tabella sono seguiti da () senza specificare le dimensioni all'interno delle parentesi;

> *tipo*, se presente, individua il tipo (INTEGER, LONG, ...) della variabile cui si riferisce; in alternativa il tipo può essere individuato dall'ultimo carattere del nome o da una precedente istruzione dichiarativa DEF*tipo*.

Nelle procedure di tipo SUB le variabili presenti in *listapar* rappresentano sia i dati su cui operare che le celle in cui verranno restituiti i risultati della procedura.
Nelle procedure di tipo FUNCTION invece, le variabili presenti individuano normalmente solo i dati su cui la funzione deve lavorare per determinare il risultato da restituire in *nomefun*.

5.15.2 Chiamata di una procedura

Una procedura SUB viene chiamata mediante l'istruzione

CALL *nomesub* [(*listaparattuali*)]

Una procedura FUNCTION viene chiamata, come per le funzioni intrinseche, facendo comparire tra gli operandi dell'espressione di una frase di assegnazione il suo nome seguito dalla *listaparattuali* posta tra parentesi.

In entrambi i casi *listaparattuali* è una lista di nomi di variabili semplici o con indice, espressioni o nomi di tabelle: questi ultimi seguiti dai simboli () senza specificare all'interno delle parentesi le dimensioni della tabella; i parametri sono separati da virgole e devono concordare in numero e tipo con i parametri formali presenti nella definizione della procedura.

Quando in un programma compare una frase di chiamata di una procedura, si passa ad eseguire le istruzioni della procedura. Al termine, l'esecuzione riprende dall'istruzione successiva a quella di chiamata.

Normalmente la procedura ha termine quando si incontra la frase END {SUB | FUNCTION}. Se sono previste uscite in punti intermedi del corpo della procedura, queste devono essere effettuate mediante l'istruzione EXIT {SUB | FUNCTION}.

La trasmissione dei parametri dal programma chiamante alla procedura può avvenire *per indirizzo* o *per valore*.
Nel primo caso viene trasmesso alla procedura l'indirizzo di memoria del parametro e la procedura lavora sul suo contenuto, per cui ogni modifica del valore del parametro all'interno della procedura si ripercuote sul programma chiamante.

Nel secondo caso viene trasmessa alla procedura una copia del valore del parametro e le modifiche eventualmente operate dalla procedura vengono eseguite sulla copia senza cambiare il valore che il parametro ha nel programma chiamante.

La *listaparattuali* può contenere parametri trasmessi sia per indirizzo che per valore.
Ogni parametro da trasmettere per valore deve comparire nella lista racchiuso tra parentesi tonde.

Una procedura SUB può restituire risultati al programma chiamante inserendoli o nelle celle predisposte o modificando i valori di alcuni parametri attuali di chiamata. Se tutti i parametri sono trasmessi per valore, la SUB quindi non restituisce risultati utilizzabili dal programma chiamante.

Una procedura FUNCTION restituisce sempre al programma chiamante il valore contenuto nella variabile *nomefun*. Eventualmente può restituire altri risultati modificando i parametri trasmessi per indirizzo.

5.15.3 Dichiarazione delle procedure usate

Quando in un modulo compaiono chiamate ad una o più procedure, ciascuna di queste deve essere dichiarata in testa al modulo tramite l'istruzione DECLARE.
In realtà l'utente è tenuto a mettere questa frase solo per FUNCTION definite in altri moduli, perché in tutti gli altri casi il QuickBASIC la inserisce automaticamente all'atto del salvataggio del programma.

La sintassi dell'istruzione è la seguente:

DECLARE {FUNCTION|SUB} {*nomefun* | *nomesub*} [(*listapar*)]

ove *listapar* ha la sintassi definita nel paragrafo 5.15.1.
Se questa lista è presente, viene controllato che i parametri di chiamata concordino in numero e tipo con i parametri inseriti nella frase DECLARE.

Esempio:
l'esempio presentato riassume alcuni dei concetti visti nei paragrafi precedenti. Si tratta di un programma che, lette due frazioni con numeratore e denominatore positivi, le riduce ai minimi termini, ne calcola la somma ed infine riduce ai minimi termini anche il risultato. È costituito da un programma principale e da due procedure, rid di tipo SUB e mcd di tipo FUNCTION, che modificano entrambe i parametri di chiamata.

Si noti che la procedura rid viene chiamata trasmettendo i parametri per indirizzo, poiché al rientro nel programma principale si utilizzano i valori modificati.
Viceversa a mcd i parametri vengono trasmessi per valore, poiché nel programma chiamante essi devono rimanere inalterati.

```
DECLARE FUNCTION mcd% (n%, m%)
DECLARE SUB rid (x%, y%)
REM Somma di due frazioni ridotte ai minimi termini
' a/b+c/d
     DEFINT A-Z
     INPUT a, b, c, d
     PRINT a; "/"; b; "+"; c; "/"; d; "=";
     CALL rid(a, b)
     CALL rid(c, d)
     PRINT a; "/"; b; "+"; c; "/"; d; "=";
     bd = mcd((b), (d))
     num = a * d / bd + c * b / bd
     den = b * d / bd
     CALL rid(num, den)
     PRINT num; "/"; den
     END
'
     FUNCTION mcd (n, m)
       WHILE n <> m
         IF n > m THEN n = n - m ELSE m = m - n
       WEND
       mcd = n
     END FUNCTION
```



```
'
     SUB rid (x, y)
       z = mcd((x), (y))
       x = x / z: y = y / z
     END SUB
```

5.15.4 Confronto tra procedure, sottoprogrammi e funzioni

I sottoprogrammi richiamati con GOSUB e le funzioni definite mediante {DEF FN | DEF FN...END DEF} hanno delle caratteristiche diverse dalle procedure definite nel paragrafo 5.15.1.
Le principali differenze sono le seguenti:

- i sottoprogrammi e le funzioni possono essere usati solo nello stesso modulo in cui sono definiti;
- in una funzione i parametri di chiamata sono trasmessi esclusivamente per valore; perciò una loro modifica nel corpo della funzione non cambia il valore all'esterno;
- tutte le variabili presenti nel corpo di un sottoprogramma o di una funzione a più frasi e distinte dai parametri di chiamata sono *globali*, cioè note a tutto il modulo; pertanto qualunque modifica del valore di una variabile nel sottoprogramma o funzione influenza la variabile in tutto il modulo.

A causa di queste caratteristiche l'uso ripetuto di un sottoprogramma per operare su dati differenti diventa oneroso; infatti è necessario, prima di ogni chiamata, copiare i dati nelle variabili globali su cui il sottoprogramma opera e, al ritorno, prelevare dalle variabili globali i risultati ricopiandoli nelle celle utilizzate in seguito.

Esempio:
il programma del paragrafo 5.15.3 è qui riscritto facendo uso di un sottoprogramma e di una funzione.

```
REM Somma di due frazioni ridotte ai minimi termini
' a/b+c/d
     DEFINT A-Z
     DEF fnmcd (n, m)
       WHILE n <> m
         IF n > m THEN n = n - m ELSE m = m - n
       WEND
       fnmcd = n
     END DEF
     INPUT a, b, c, d
     PRINT a; "/"; b; "+"; c; "/"; d; "=";
     x = a: y = b: GOSUB rid: a = x: b = y
     x = c: y = d: GOSUB rid: c = x: d = y
     PRINT a; "/"; b; "+"; c; "/"; d; "=";
     bd = fnmcd(b, d)
     num = a * d / bd + c * b / bd
     den = b * d / bd
     x = num: y = den: GOSUB rid: num = x: den = y
     PRINT num; "/"; den
     END
rid:
     z = fnmcd(x, y)
     x = x / z: y = y / z
     RETURN
```

5.15.5 Ricorsione

Le procedure Basic possono essere ricorsive, cioè possono richiamare se stesse direttamente o indirettamente. È opportuno però utilizzare la ricorsione solo quando non esistono procedure di calcolo alternative. I programmi ricorsivi sono infatti lenti ed occupano tanta più memoria quanto più è "profonda" la ricorsione.

Come esempio si riporta il programma per il calcolo del coefficiente binomiale secondo le due definizioni:

$$\binom{n}{m} = \frac{n(n-1)\cdots(n-m+1)}{m(m-1)\cdots 1}$$

$$\binom{n}{m} = \binom{n-1}{m-1} + \binom{n}{m-1}$$

```
DECLARE FUNCTION nsm& (a&, b&)
' Coefficiente binomiale con formula diretta o ricorsiva
   DEFLNG I-N
   INPUT "n,m ", n, m
   PRINT " n  m   nsum  sec.(diretto)  sec.(ricorsivo)"
   PRINT
   WHILE (n <> 0 AND m <> 0)
     PRINT USING "## ##"; n; m;
     z = TIMER: k = 1
     FOR i = 1 TO m
       k = k * (n - i + 1) / i
     NEXT
     z1 = TIMER: PRINT USING " ######"; k;
     PRINT SPC(6); z1 - z,
     z = TIMER
     k = nsm((n); (m))
     z1 = TIMER
     PRINT SPC(4); z1 - z
     INPUT "n,m ", n, m
   WEND
   END
'
   DEFLNG A-H
   FUNCTION nsm (a, b)
     IF a < 0 OR a < b THEN PRINT "errore": EXIT FUNCTION
     IF a = b OR b = 0 THEN nsm = 1: EXIT FUNCTION
     nsm = nsm(a - 1, b - 1) + nsm(a - 1, b)
   END FUNCTION
```

Il programma stampa anche i tempi di calcolo nei due algoritmi.

La tabella seguente riporta i risultati per diversi valori di n e m; come si può notare il calcolo ricorsivo diventa sempre più lento al crescere di n.

n	m	nsum	sec.(diretto)	sec.(ricorsivo)
6	3	20	0	0
8	4	70	0	5.859375E-02
9	5	126	0	5.078125E-02
10	5	252	0	5.078125E-02
12	6	924	0	.1132813
14	7	3432	0	.4921875
16	8	12870	0	1.867188
18	9	48620	0	7.03125
20	10	184756	0	26.75

5.16 Gestione degli errori

Il verificarsi di un errore durante l'esecuzione di un programma interrompe l'esecuzione stessa e provoca la comparsa sul video di un messaggio che segnala il tipo di evento e individua la porzione del programma in cui si trova l'istruzione che lo ha generato.

Per evitare tali interruzioni l'utente può assumere il controllo degli errori e, a seconda del tipo di evento, far eseguire particolari istruzioni (*routine di trattamento dell'errore*).

L'istruzione

ON ERROR GOTO *etic*

fa sì che il verificarsi di un errore nella linee di programma ad essa successive trasferisca il controllo dell'esecuzione alla istruzione con etichetta *etic*, che deve appartenere allo stesso modulo ed è la prima di un insieme di istruzioni che specificano l'azione correttiva.

La routine di trattamento dell'errore deve terminare o con l'istruzione END, per interrompere definitivamente il programma, o con RESUME per riprenderlo.

La frase RESUME può presentarsi in differenti forme:

RESUME NEXT

riprende l'esecuzione dall'istruzione successiva a quella che ha provocato l'interruzione;

RESUME [0]

riprende dalla frase che ha causato l'errore;

RESUME *etic*

riprende dalla linea con etichetta *etic*.

La routine di trattamento dell'errore può prendere azioni correttive diverse a seconda del tipo di errore utilizzando le funzioni senza parametri ERR e ERL, che restituiscono rispettivamente il codice numerico dell'errore ed il numero di linea in cui è avvenuto.
Il valore di ERL è significativo solo se il programma contiene etichette numeriche: in tal caso la funzione restituisce l'ultimo numero di linea precedente a quella che ha causato l'errore.
Se il programma non contiene etichette numeriche, ERL restituisce sempre il valore 0.

5.16.1 Messaggi e codici di errore

Sono qui elencati i codici numerici degli errori più comuni individuabili all'atto dell'esecuzione, con il loro significato.

3 `RETURN without GOSUB`

Tentativo di ritorno da un sottoprogramma eseguito senza un trasferimento mediante GOSUB.

4 `Out of DATA`

Tentativo di eseguire una READ senza più dati disponibili in frasi DATA.

5 `Illegal function call`

Chiamata di funzione con valori illegali dei parametri; l'errore può comparire anche in altri casi, per esempio per uso di istruzioni grafiche in modo testo o per elevamento di un numero negativo a potenza non intera.

6 `Overflow`

Il risultato del calcolo è al di fuori dell'intervallo di valori permessi per il tipo di variabile in questione.

9 `Subscript out of range`

Valore di un indice di tabella al di fuori delle dimensioni dichiarate.

11 `Division by zero`

Divisione per zero.

19 `No RESUME`

L'esecuzione di una routine d'errore termina senza aver trovato una END o una RESUME.

20 `RESUME without error`

Tentativo di eseguire una RESUME al di fuori di una routine d'errore.

27 `Out of paper`

Manca la carta nella stampante.

52 `Bad file name or number`

Una istruzione fa riferimento a un file con un nome o un numero non precedentemente specificato in una OPEN.

53 `File not found`

Il file indicato in una OPEN non esiste.

54 `Bad file mode`

Uso di PUT o GET su un file sequenziale oppure tentativo di lettura da file aperto per scrittura o viceversa.

55 `File already open`

Tentativo di aprire un file già aperto.

59 `Bad record length`

Tentativo di eseguire una PUT o GET che specifica un record di lunghezza diversa da quella specificata nella OPEN del file.

62 `Input past end of file`

Tentativo di leggere oltre la fine del file.

63 `Bad record number`

Numero di record negativo o nullo in una istruzione PUT o GET.

64 `Bad file name`

Il nome di file indicato in una OPEN non è sintatticamente corretto.

71 `Disk not ready`

Non c'è dischetto nell'unità oppure l'unità è aperta.

76 `Path not found`

Il pathname specificato in una OPEN non esiste.

5.17 Esecuzione di comandi MS-DOS

Sono disponibili in Basic alcune istruzioni che permettono di compiere da programma operazioni su files e direttori, di eseguire qualsiasi altro comando MS-DOS oppure di uscire dall'ambiente Basic.

In tutte le istruzioni che seguono, i campi variabili possono essere sia costanti stringa che espressioni di tipo stringa aventi il valore desiderato.

5.17.1 Operazioni sui files

KILL *nomefile*

cancella il file con pathname *nomefile*; è analoga al comando DEL di MS-DOS.

Esempi:
```
KILL "*.BAS"
F$ = "A:\DATI\D1.DAT":  KILL F$
```

NAME *nome-vecchio* AS *nome-nuovo*

è analoga al comando REN di MS-DOS. In aggiunta, può spostare un file da un direttorio ad un altro dello stesso disco.

Esempi:
```
NAME "D1.DAT" AS "DATI.DAT"
NAME "D1.DAT" AS "\DATI.DAT"
```

5.17.2 Operazioni sui direttori

Le tre istruzioni

MKDIR *nome-dir*

RMDIR *nome-dir*

CHDIR *nome-dir*

sono analoghe rispettivamente ai comandi MD, RD e CD di MS-DOS.

5.17.3 Altri comandi al sistema

SHELL *comando*

provoca l'uscita dall'ambiente Basic, l'esecuzione del comando MS-DOS indicato nell'istruzione e il ritorno al programma a partire dall'istruzione successiva alla SHELL.

Esempi:
```
SHELL "PRINT NOTE.TXT"
COM$ = "DIR ":  D$ = "C:\USER":  SHELL COM$ + D$
```

5.17.4 Rientro al sistema operativo

L'istruzione

SYSTEM

chiude tutti i files aperti e ritorna il controllo al sistema operativo.
Per vedere un esempio d'uso dell'istruzione SYSTEM si esamini il programma di gestione del menu principale del software ESERCIZI, che si trova in C:\MATR\ESERCIZI.BAS e permette l'uscita automatica dall'ambiente QuickBASIC alla fine dell'utilizzo.

Capitolo 6

L'ambiente QuickBASIC

QuickBASIC è un ambiente di programmazione *integrato*, strutturato a menu, che mette a disposizione tutti gli strumenti necessari per predisporre un programma in linguaggio Basic, eseguirlo e correggerlo.

Dall'ambiente MS-DOS si accede a QuickBASIC mediante il comando

QBX

All'utente si presenta uno schermo suddiviso in due finestre (vedi figura 6.1): la *finestra di testo* nella parte superiore e la *finestra immediata* in quella inferiore.

Nella prima linea dello schermo (*riga dei menu*) compaiono i nomi dei menu, ciascuno dei quali individua un insieme di comandi disponibili. L'ultima riga (*riga di stato*) contiene di solito alcune informazioni sul contesto attuale.

Il cursore inizialmente si trova sulla finestra di testo. Il tasto *F6* permette di spostarlo, ciclicamente, dall'una all'altra delle due finestre presenti sullo schermo.

6.1 Finestra immediata

La finestra immediata permette di digitare una linea di istruzioni Basic che viene eseguita non appena premuto il tasto *Enter*.
Può contenere fino a 10 linee, in relazione o no l'una con l'altra. Ci
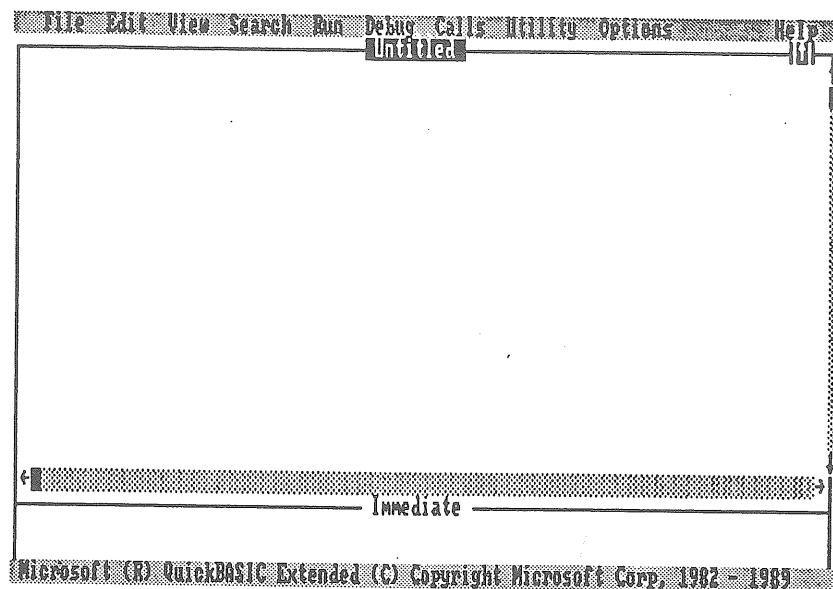
Figura 6.1

si sposta nella linea con i tasti freccia. Si preme *Enter* per eseguire la linea su cui si trova il cursore.

La finestra immediata può servire sia per fare calcoli immediati o prove di esecuzione di frasi di programma che per visualizzare o cambiare il contenuto di variabili del programma presente nella finestra testo e in corso di esecuzione; in questo caso si interrompe il programma, si passa nella finestra immediata e si controllano ed eventualmente si cambiano i valori di alcune variabili, poi si riprende il programma interrotto.

Nella finestra immediata non sono permesse le frasi la cui sintassi si estende su più linee successive o le frasi dichiarative (DIM, REDIM, DEF*tipo*).

6.2 Finestra di testo: "editing" del programma sorgente

All'entrata nell'ambiente QuickBASIC il cursore si trova nella finestra di testo e l'utente può scrivere direttamente il proprio programma.
Il nome del programma compare in alto nella *riga titolo*; se il programma è nuovo, la riga contiene la dizione `< Untitled >`.

Viene attivato automaticamente un editore di testi che svolge le seguenti operazioni non appena l'utente immette una linea di programma (terminandola con *Enter* o spostando il cursore su un'altra linea con un tasto freccia):

- controlla la correttezza sintattica della linea;

- riscrive la linea in formato omogeneo, introducendo le opportune spaziature e mettendo in maiuscolo le parole chiave; inoltre tiene traccia dei nomi delle variabili e delle procedure e fa sì che ogni nome compaia in tutto il programma sempre in caratteri maiuscoli o minuscoli eguali a quelli con cui è stato introdotto nell'ultima linea immessa;

- se la linea è corretta, la traduce in modo che sia pronta per l'esecuzione.

In caso di errore sintattico, viene aperta una finestra che specifica il tipo di errore. Premendo *Esc* o la barra di spaziatura la finestra scompare e il cursore si posiziona sulla linea errata. Spostando il cursore sulla parola chiave e premendo *F1* si ottengono informazioni sulla sintassi dell'istruzione.
Il controllo sintattico effettuato dall'editore non riconosce certi tipi di errori, che verranno perciò segnalati solo in fase di esecuzione del programma.

Durante la stesura del testo normalmente si è in *modo inserimento*: il cursore si presenta come una lineetta e i caratteri battuti vengono inseriti sullo schermo nella posizione del cursore traslando a destra quelli eventualmente seguenti.

Nel *modo sovrascrittura* il cursore si presenta come un quadratino pieno e il carattere battuto rimpiazza quello presente nella posizione del cursore. Si passa dall'uno all'altro modo premendo il tasto *Insert*.

Il cursore può essere mosso in qualunque punto del testo, per consentire inserimenti e correzioni, mediante i seguenti tasti:

←, →, ↑, ↓	un carattere a sinistra, a destra, in su, in giù;
Home	inizio riga;
End	fine riga;
Ctrl Home	inizio testo;
Ctrl End	fine testo;
Ctrl Enter	inizio della riga successiva.

Le cancellazioni di uno o più caratteri di una riga si effettuano con:

Delete	carattere su cui sta il cursore;
Backspace	carattere che precede il cursore;
Ctrl Y	riga corrente;
Ctrl Q Y	dal cursore a fine riga.

Per cancellare, spostare o duplicare parti di testo, occorre prima selezionare la parte voluta. Ci si posiziona col cursore sul primo carattere del testo da selezionare; si evidenzia poi tutto il testo che deve essere scelto tenendo premuto il tasto *Shift* e muovendosi contemporaneamente con le frecce. Successivamente:

Delete	cancella il testo selezionato;
Shift Delete	cancella il testo selezionato e lo pone in una memoria temporanea;
Ctrl Insert	copia il testo selezionato in una memoria temporanea senza cancellarlo dalla posizione attuale;
Shift Insert	dopo aver spostato il cursore nella posizione voluta, inserisce il testo precedentemente memorizzato. Il testo viene spostato o copiato nella nuova posizione, a seconda che il comando precedente sia stato *Shift Delete* o *Ctrl Insert*. Se il comando precedente è stato *Ctrl Y*, *Shift Insert* inserisce la riga cancellata.

6.2.1 "Editing" di un modulo costituito da più procedure

Se il programma sorgente contiene una o più procedure, la sua stesura può essere effettuata in uno o più moduli (vedi paragrafo 5.15).

Se il modulo è uno solo, quando l'utente, dopo aver scritto il programma principale, introduce una istruzione **{SUB | FUNCTION}** seguita dal nome di una procedura, QuickBASIC si comporta nel seguente modo:

- fa scomparire dalla finestra di testo il contenuto precedente pur mantenendolo in memoria;
- fa comparire le due istruzioni **{SUB...END SUB | FUNCTION...END FUNCTION}**;
- fa precedere queste istruzioni da una copia delle eventuali istruzioni **DEF***tipo* presenti nel programma principale. Si faccia attenzione che queste istruzioni non vengono aggiornate automaticamente quando l'utente modifica le stesse presenti nel programma principale o ne aggiunge di nuove; in questo caso la modifica in testa alle procedure è quindi a carico dell'utente.

Nella finestra di testo l'utente immette il corpo della procedura. La riga del titolo mostra il nome del modulo seguito dal nome della procedura che si sta predisponendo.
Al momento del salvataggio del modulo, QuickBASIC inserisce in testa allo stesso una frase **DECLARE** per ogni procedura richiamata.

Se il programma sorgente è costituito da più moduli, nella stesura di un modulo contenente solo procedure, QuickBASIC all'atto della scrittura di una istruzione **{SUB | FUNCTION}** si comporta analogamente a quanto detto sopra: in questo caso viene creata una parte principale che inizialmente risulta vuota e nella quale, al momento del salvataggio, vengono inserite le frasi **DECLARE** relative alle procedure di tipo **FUNCTION** definite nel modulo e a quelle di tipo **SUB** da esse eventualmente richiamate.

Nei moduli contenenti solo procedure, le frasi **DEF***tipo* devono essere poste dall'utente in concordanza con quelle da lui immesse nel programma principale.
Inoltre se da un modulo si richiamano funzioni definite in altri moduli, le corrispondenti **DECLARE** devono essere inserite dall'utente in testa al modulo contenente il programma principale.

Nel paragrafo 6.8.2 verrà spiegato dettagliatamente come, mediante il menu **View**, si possano trattare le varie procedure.

6.3 Scelta di menu e comandi

I menu disponibili nell'ambiente QuickBASIC sono i seguenti:

File	gestione di files sorgente e uscita da QBX
Edit	correzione del testo
View	visualizzazione di diverse finestre
Search	ricerche nel testo
Run	esecuzione di programmi e gestione di librerie
Debug	controllo dell'esecuzione di un programma
Calls	non contiene comandi; dà informazioni sulle chiamate tra le procedure del programma
Utility	esecuzione di comandi MS-DOS
Options	gestione dei parametri dell'ambiente QBX
Help	aiuto sulla sintassi e sul significato delle istruzioni e delle funzioni Basic

I menu **Calls** e **Options** non verranno trattati; **Options** riguarda la configurazione del QuickBASIC che per i personal del Laboratorio è stata definita all'atto dell'installazione e non deve essere modificata, **Calls** è poco utilizzato per le esercitazioni del corso.

Per scegliere un menu è necessario procedere come segue:

1) accedere alla riga menu premendo il tasto *Alt*;

2) premere la lettera evidenziata nel nome del menu desiderato; oppure spostarsi con le frecce ← e → sul nome desiderato evidenziandolo e premere *Enter*.

In seguito alla scelta del menu, viene mostrata la lista dei comandi ad esso associati.

Se, per sbaglio, si è scelto un menu diverso da quello voluto, si può battere *Esc* per cancellare l'operazione e rieseguirla dal punto 1); in alternativa, ci si può spostare con le frecce ← e → sul menu desiderato.

Per eseguire uno dei comandi presenti nella lista si preme il tasto corrispondente alla lettera evidenziata oppure ci si sposta con le frecce ↑ e ↓ sul comando evidenziandolo e si preme *Enter*.

Alcuni comandi possono essere eseguiti anche senza accedere ai menu semplicemente battendo una certa combinazione di tasti (*forma abbreviata del comando*).

Nella lista dei comandi la dicitura può essere seguita da tre punti (...) o dall'indicazione della forma abbreviata del comando.
I comandi non seguiti dai tre punti vengono eseguiti non appena scelti, gli altri richiedono ulteriori informazioni in una *finestra di dialogo*, che si presenta diversamente a seconda del comando.
In generale una finestra di dialogo è composta da più sottofinestre. Ci si muove da una all'altra con *Tab* (in avanti) o *Shift Tab* (all'indietro) oppure ci si posiziona direttamente con *Alt* e la lettera evidenziata nella sottofinestra.
All'interno di una sottofinestra ci si muove con le frecce.

Nella parte bassa della finestra di dialogo ci sono sempre due o più scelte di azioni da intraprendere in base alle informazioni fornite; le tre scelte quasi sempre presenti sono `< OK >` e `< Cancel >`, che rispettivamente attivano e cancellano il comando, e `< Help >` che dà un aiuto nella compilazione della finestra.
Quando si batte *Enter* viene eseguita l'azione scelta. Una delle azioni (di solito `< OK >`) è sempre racchiusa tra parentesi evidenziate; la sua scelta non richiede il posizionamento su di essa mediante il cursore.
Per annullare il comando si può scegliere `< Cancel >` o più semplicemente premere *Esc*.

6.3.1 Aiuto sul significato dei comandi

Nel seguito verranno illustrati solo alcuni comandi dei menu di uso più frequente. Gli utenti possono comunque avere informazioni sullo scopo e sull'utilizzo di ogni menu e comando e sulla compilazione delle finestre di dialogo.

Per avere informazione su un menu, se ne evidenzia il nome sulla riga dei menu e si preme *F1*.

Per avere informazioni su un comando, se ne evidenzia il nome sulla lista dei comandi del menu corrispondente e si preme *F1*.

Per aiuto nella compilazione di una finestra di dialogo, si sceglie l'azione < `Help` > nella finestra stessa.
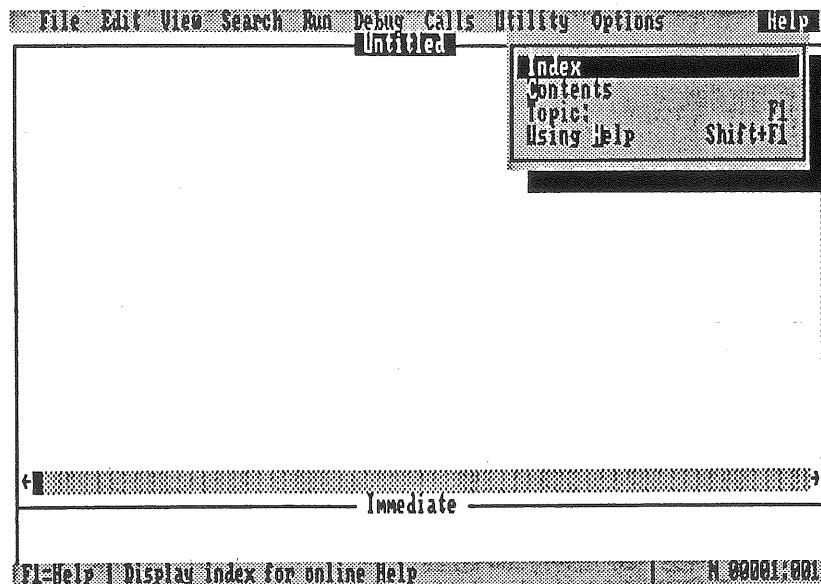
Figura 6.2

6.4 Informazioni sulle istruzioni Basic: menu Help

Tra i comandi del menu **Help** (vedi figura 6.2), **Using Help** (o la sua forma abbreviata *Shift F1*) fornisce le informazioni per un uso corretto del sistema di aiuto disponibile in linea. Di seguito si daranno pertanto solo le informazioni essenziali.
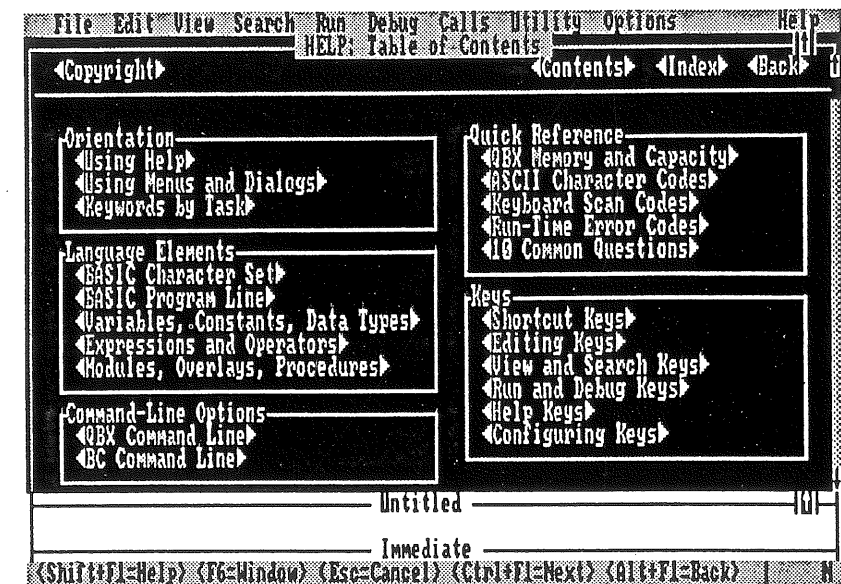
Figura 6.3

I comandi **Contents** e **Index** presentano rispettivamente un sommario di argomenti dell'Help e un indice analitico alfabetico e forniscono informazioni generali sul linguaggio e sull'ambiente, indipendentemente dal programma in corso.

Quando si sceglie il comando **Index** compare la finestra di dialogo di figura 6.3 e quindi:

- si digita la lettera iniziale dell'argomento su cui si vogliono spiegazioni e si preme *F1*: vengono visualizzati tutti gli argomenti che hanno l'iniziale scelta;

- si preme ripetutamente l'iniziale fino a che il cursore si posiziona sull'argomento desiderato o, in alternativa, ci si sposta usando le frecce; in entrambi i casi si preme poi *F1*.
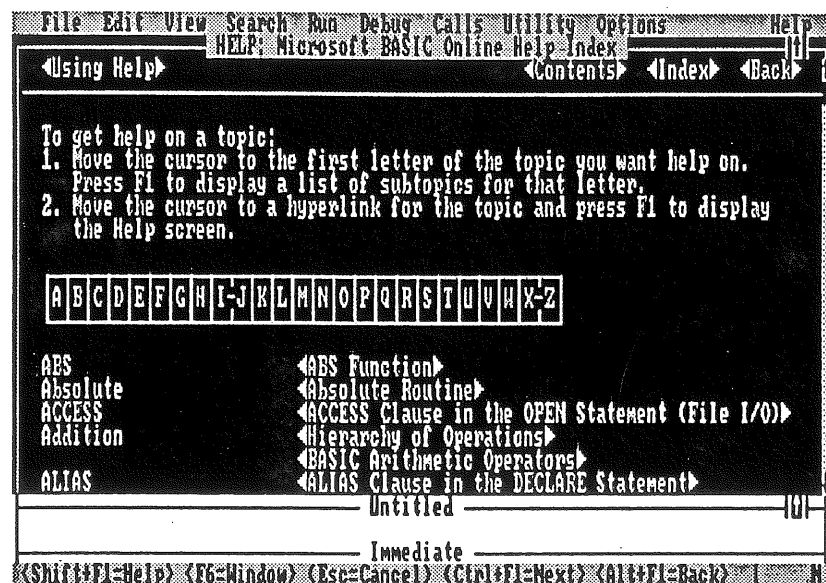
Figura 6.4

Quando si sceglie il comando **Contents** compare il sommario di figura 6.4. Premendo ripetutamente l'iniziale dell'argomento voluto il cursore si sposta sui vari campi aventi la stessa iniziale, fino a che ci si trova su quello desiderato. In alternativa ci si può posizionare premendo ripetutamente *Tab*. In entrambi i casi si preme poi *F1*.

Il comando **Topic** può essere di grande utilità durante la stesura di un programma per avere informazioni sull'oggetto (parola chiave, nome di variabile o di procedura) puntato dal cursore nella finestra di testo. Il comando è usato più comunemente nella sua forma abbreviata *F1*.

Se il cursore è su una parola chiave, viene fornita la sintassi delle istruzioni in cui essa può comparire.

Se è posizionato su un nome di variabile, vengono visualizzati il tipo della variabile e il pathname del modulo in cui essa compare.

Se è sul nome di una procedura, vengono fornite le informazioni relative.

6.5 Gestione di files sorgente: menu File

I comandi del menu **File** (vedi figura 6.5) permettono il caricamento in memoria, la memorizzazione su disco e la stampa dei files sorgente.
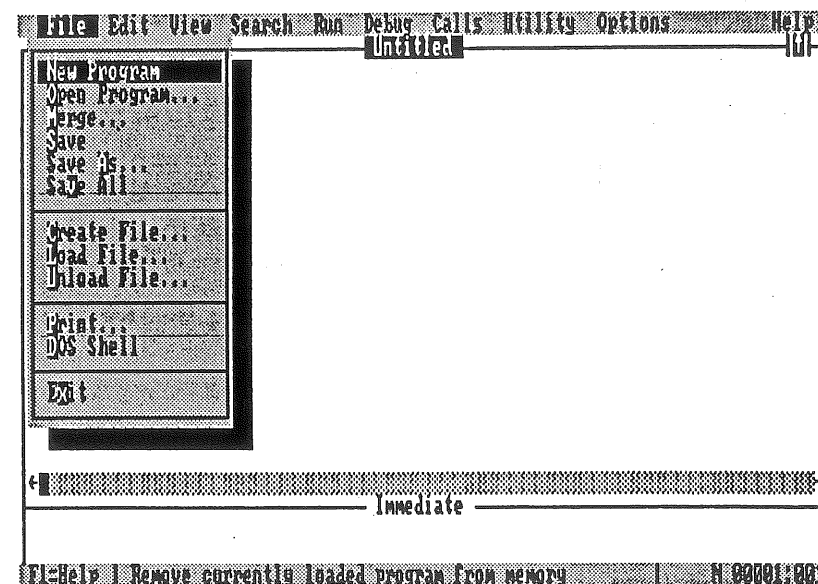
Figura 6.5

Possono agire su un programma composto da uno o più moduli.

6.5.1 Salvataggio di un programma su disco

I comandi **Save** e **Save As** permettono di salvare su disco il modulo corrente, il cui nome compare nella riga del titolo.

Se il modulo non ha ancora nome, i due comandi sono equivalenti; viene aperta la finestra di dialogo di figura 6.6 e il cursore è posizionato sulla sottofinestra `File Name` in attesa del nome da attribuire al file: l'estensione BAS può essere omessa perché aggiunta automaticamente.

La riga sottostante evidenzia il nome del direttorio corrente, dove il file verrà memorizzato a meno di diversa indicazione; questa può essere fornita o dando come nome del file il pathname completo o scegliendo nella sottofinestra `Dirs/Drives` il disco e il direttorio voluti.
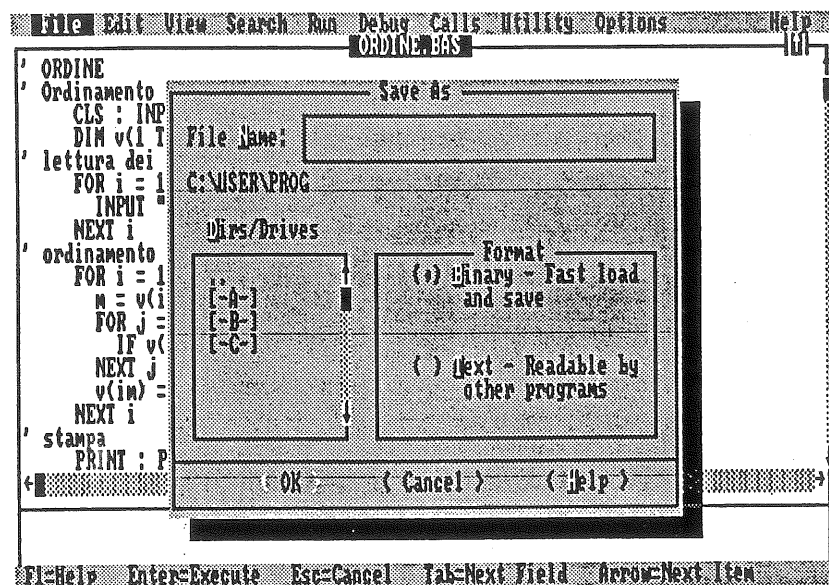
Figura 6.6

La sottofinestra `Format` consente di scegliere tra due modalità di memorizzazione: `Binary` e `Text`.
La prima crea files che vengono caricati più velocemente in ambiente QuickBASIC ma non possono essere trattati in altri ambienti.
La seconda memorizza i files in formato ASCII e li rende quindi utilizzabili da qualunque altro programma (per esempio dai comandi TYPE e PRINT di MS-DOS).
Il formato predisposto inizialmente è quello binario; per cambiarlo basta spostarsi su `Text` con la freccia ↓.

Una volta che la finestra di dialogo contiene tutte le informazioni volute, basta premere *Enter* per renderle esecutive.

Se il modulo ha già un nome, **Save** lo salva senza ulteriori richieste mentre **Save As** permette di riconfermare il nome o eventualmente di

cambiarlo.

Il comando **Save All** consente di salvare tutti i moduli distinti che compongono il programma e sono presenti in memoria. Inoltre predispone un file con il nome del programma principale ed estensione MAK che contiene i nomi di tutti i moduli e verrà utilizzato durante un successivo caricamento del programma (vedi paragrafo 6.5.3).

6.5.2 Predisposizione per un nuovo programma

Quando si entra in ambiente QuickBASIC mediante il comando **QBX** viene attivato automaticamente l'editore di testi per la scrittura di un nuovo programma.
Se si è già in ambiente QuickBASIC e ci sono moduli caricati, per cancellarli dalla memoria e iniziare la scrittura di un nuovo programma si usa il comando **New Program**. Se alcuni moduli non sono ancora stati salvati, **New Program** lo segnala e ne permette la memorizzazione.

6.5.3 Caricamento di un programma

Il comando **Open Program** legge da disco e porta in memoria un modulo cancellando gli eventuali moduli già presenti. La scelta di **Open Program** fa aprire una finestra di dialogo (vedi figura 6.7).
Come per il comando **Save As**, il cursore è posizionato sulla sottofinestra `File Name` in attesa del nome del file da caricare e la riga sottostante evidenzia il direttorio corrente; i files di tale direttorio sono elencati nella sottofinestra `Files`.

L'utente può scrivere il nome del file desiderato nella sottofinestra `File Name` oppure può spostarsi nella sottofinestra `Files` e posizionarsi con le frecce sul file voluto; la pressione del tasto *Enter* rende esecutivo il comando.
Se il programma è costituito da più moduli, il comportamento del comando dipende dalla presenza nel direttorio del file con nome eguale a quello del modulo principale ed estensione MAK.
Se esso è presente, caricando con **Open Program** il modulo principale vengono automaticamente caricati tutti i moduli costituenti il
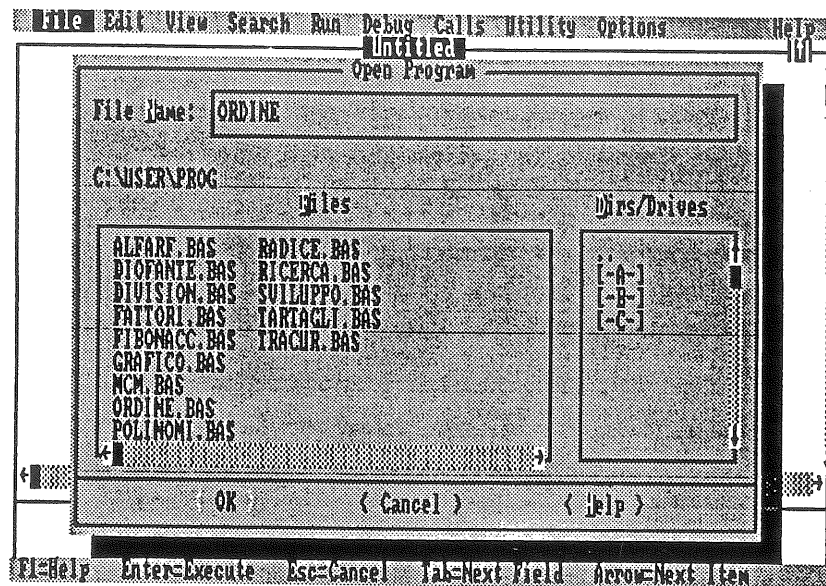
Figura 6.7

programma.
In assenza del file .MAK conviene caricare con **Open Program** il modulo principale e con **Load File** tutti gli altri moduli.

Il comando **Load File** apre una finestra di dialogo analoga a quella aperta da **Open Program**. All'atto del caricamento di ogni modulo, ne viene mostrata nella finestra di testo la parte principale che, nel caso in cui il modulo sia costituito solo da procedure, contiene solo frasi dichiarative o è vuota.

6.5.4 Stampa di un programma

Il comando **Print** apre una finestra di dialogo col cursore posizionato nella sottofinestra **Send**, dove si può scegliere se stampare il testo selezionato, il contenuto della finestra attiva, il modulo corrente o tutti quelli presenti in memoria. La scelta predisposta stampa il modulo corrente presente nella finestra di testo e può essere modificata con l'uso delle frecce.

Non si devono fare modifiche alla finestra To in quanto già predisposta per l'uscita sulla stampante collegata alla porta parallela.

6.5.5 Uscita dall'ambiente QuickBASIC

Il comando **Dos Shell** fa uscire temporaneamente dall'ambiente per consentire all'utente l'uso di comandi del sistema operativo. QuickBASIC rimane caricato in memoria e si rientra nell'ambiente mediante il comando EXIT dato dal prompt di MS-DOS.

Il comando **Exit** del menu **File** fa uscire definitivamente dall'ambiente QuickBASIC e tornare a MS-DOS. Se a questo punto ci sono in memoria alcuni moduli che non sono ancora stati salvati il comando lo segnala e ne permette la memorizzazione.

6.6 "Editing" del testo: menu Edit

Il menu **Edit** (vedi figura 6.8) comprende i comandi **Cut**, **Copy**, **Paste** e **Clear** che sono già stati spiegati nelle loro forme abbreviate nel paragrafo 6.2.

I comandi **New SUB** e **New FUNCTION** hanno lo stesso effetto della scrittura delle frasi **SUB** e **FUNCTION** nella finestra di testo (vedi paragrafo 6.2.1).

I comandi **Undo** e **Redo** agiscono sulla linea su cui si stanno facendo modifiche.
Se il cursore viene spostato dalla linea, le modifiche fatte diventano permanenti, altrimenti il comando **Undo** riporta alla sua forma iniziale la linea su cui sono state fatte modifiche, mentre **Redo** compie l'operazione opposta ripristinando le modifiche eventualmente cancellate con **Undo**.
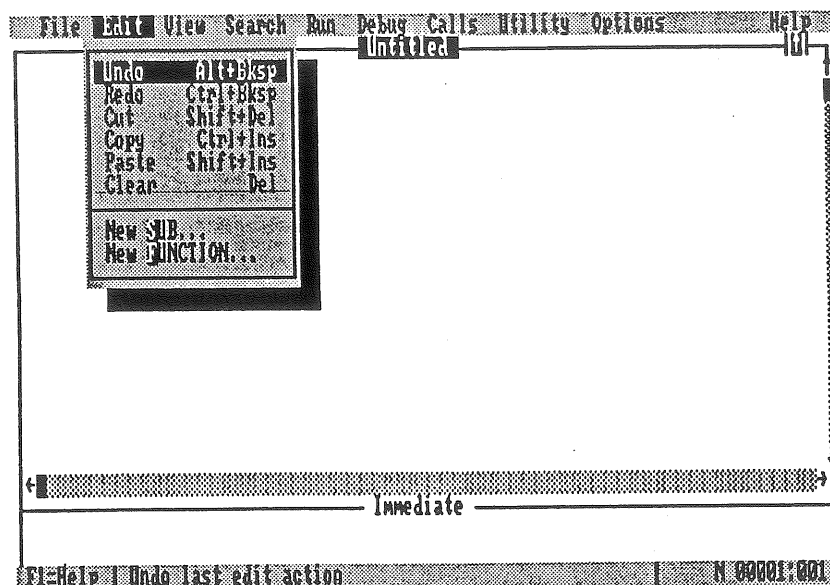
Figura 6.8

6.7 Ricerca di stringhe nel testo: menu Search

Il menu **Search** è composto dai comandi presentati in figura 6.9.

Il comando **Find** cerca nella finestra attiva, nel modulo corrente o in tutti quelli caricati, la stringa di caratteri che l'utente ha fornito nella finestra di dialogo associata.
Se la stringa è presente, essa viene evidenziata e il cursore è posizionato sul suo primo carattere.

Per ripetere la ricerca si usa il comando **Repeat Last Find** o la sua forma abbreviata *F3*.

Il comando **Change** permette di sostituire le occorrenze di una stringa con un'altra.
L'utente fornisce le due stringhe nella finestra di dialogo e può scegliere se effettuare la sostituzione in modo automatico o con conferma prima di ogni modifica.
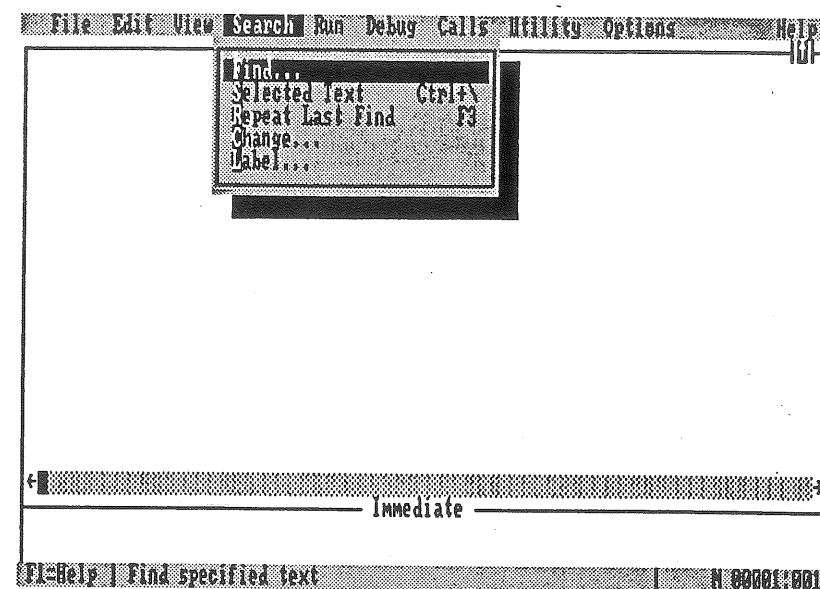
Figura 6.9

6.8 Menu View

Il menu **View** (vedi figura 6.10) permette di visualizzare la *finestra dei risultati* (*output screen*), di cambiare il contenuto della finestra di testo e di gestire moduli e procedure.

6.8.1 Visualizzazione della finestra dei risultati

La finestra dei risultati è quella in cui vengono inseriti i dati e stampati i risultati durante l'esecuzione di un programma.
Il comando **Output screen**, utilizzato più frequentemente con la forma abbreviata *F4*, consente di visualizzare questa finestra per esaminarne il contenuto in qualunque momento della sessione QuickBASIC.
Più precisamente, si passa dalla finestra di testo a quella dei risultati o viceversa ogni volta che si preme *F4*.
Il comando è particolarmente utile quando durante il "debugging" di un programma si vogliono esaminare via via i risultati intermedi (vedi paragrafo 6.11).

Figura 6.10

6.8.2 Gestione delle procedure

I comandi **SUBs** e **Split** sono usati solo se il programma caricato in memoria è costituito da più procedure suddivise in uno o più moduli. Come già accennato, in questo caso nella finestra di testo compare solo la parte principale dell'ultimo modulo caricato.

Il comando **SUBs**, utilizzato più comunemente mediante la forma abbreviata *F2*, apre una finestra di dialogo in cui sono riportati i nomi di tutti i moduli costituenti il programma e, per ogni modulo, il nome di tutte le procedure che lo compongono. Il nome della procedura presente nella finestra di testo risulta evidenziato.
Mediante le frecce ci si può spostare da un nome all'altro; la riga sottostante la sottofinestra dei nomi indica il ruolo che il modulo scelto ha all'interno del programma (modulo principale, modulo, FUNCTION o SUB).
Se si preme *Enter* si rende esecutiva l'azione evidenziata `< Edit in Active >` e il testo corrispondente al nome scelto viene portato nella finestra di testo.

Se si vuole che il modulo precedente rimanga sullo schermo accanto a quello scelto, occorre scegliere l'azione `< Edit in Split >` prima di rendere esecutivo il comando mediante *Enter*. In questo caso lo schermo viene diviso orizzontalmente in due finestre di testo, contenenti i due moduli. Con *F6* ci si sposta circolarmente tra queste due finestre e quella immediata.

Nella finestra di dialogo aperta da **SUBs**, oltre alle azioni sopraddette e a quelle, sempre presenti, di `< Help >` e `< Cancel >`, ce ne sono altre due: `< Delete >` consente di scaricare dalla memoria il modulo o la procedura scelta, `< Move >` permette di spostare una procedura da un modulo all'altro.

La divisione in due della finestra di testo si può ottenere anche con il comando **Split** del menu. Il comando genera una seconda finestra con lo stesso contenuto della prima. Si può cambiare il contenuto di una delle due finestre con il comando **SUBs**.
Per riportare lo schermo alla condizione iniziale con una sola finestra di testo, basta scegliere di nuovo il comando **Split**.

6.9 Esecuzione di un programma: menu Run

I comandi **Start**, **Restart** e **Continue** del menu **Run** (vedi figura 6.11) vengono utilizzati per l'esecuzione di un programma.

Il comando **Start**, usato preferibilmente mediante la forma abbreviata *Shift F5*, inizializza tutte le variabili del programma attualmente in memoria e lancia l'esecuzione a partire dalla prima istruzione eseguibile.

Il comando **Continue**, abbreviato in *F5*, fa riprendere l'esecuzione di un programma dopo che questo è stato interrotto o a causa della presenza di una istruzione **STOP** oppure premendo i tasti *Ctrl Break*.

Il comando **Restart** reinizializza tutte le variabili del programma e lo predispone per essere eseguito a partire dalla prima istruzione. A differenza di **Start**, **Restart** non lancia l'esecuzione; il comando, come

si vedrà nel paragrafo 6.11, serve soprattutto durante il "debugging" del programma.
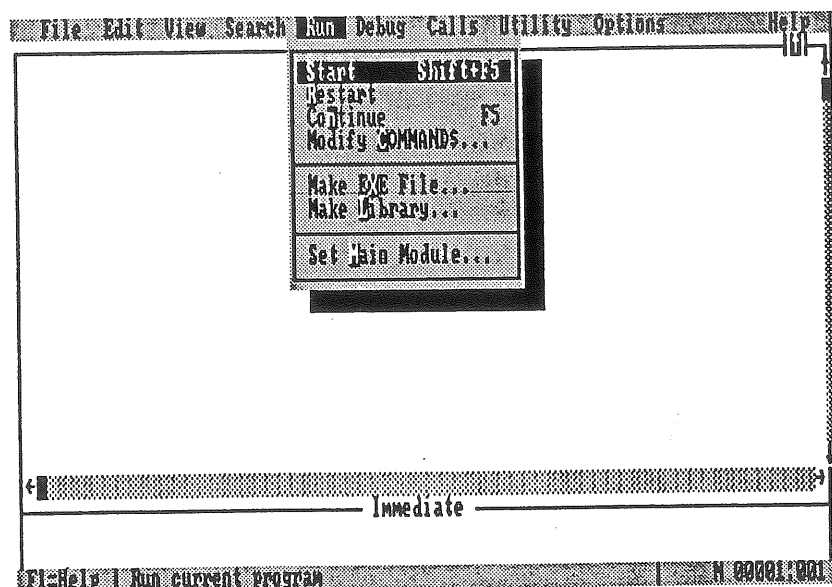
Figura 6.11

Il comando **Set Main Module** permette di scegliere tra i moduli presenti in memoria ed elencati in una finestra di dialogo quello che deve fungere da modulo principale. È particolarmente utile quando il programma è costituito da più moduli che sono stati caricati in ordine sparso.

Il comando **Make Exe File** permette di creare mediante compilazione e collegamento un file con estensione EXE contenente il codice eseguibile del programma. Tale codice ha il vantaggio di essere eseguito più velocemente e di essere portabile anche su elaboratori non dotati di QuickBASIC. Il programma in questa versione va eseguito direttamente in ambiente MS-DOS scrivendone semplicemente il nome dal prompt del sistema operativo.

La finestra di dialogo associata al comando **Make Exe File** presenta una scelta di opzioni dipendenti anche dalla configurazione hardware

dell'elaboratore usato. È opportuno quindi fornire solo il nome del file e lasciare le altre opzioni alla scelta già predisposta.

Un'altra possibilità offerta dal menu **Run** è la costruzione di librerie di procedure mediante il comando **Make Library**.
Si caricano in memoria le procedure da includere nella libreria (e solo quelle!) e si dà il comando **Make Library**. Alla richiesta di conferma del salvataggio dei files l'utente deve rispondere in modo affermativo.
Successivamente compare una finestra di dialogo in cui bisogna inserire il nome della libreria; normalmente non è necessario modificare le altre opzioni già impostate.
Il comando genera due files, di estensione rispettivamente QLB e LIB.

Per usare la libreria così costruita occorre entrare in ambiente QBX mediante il comando

QBX *nome* /L *nomelib*

ove *nome* è il nome del programma che utilizza le procedure e *nomelib* è il nome della libreria costruita.

Bisogna far attenzione che non esista nel direttorio un file *nome*.MAK perché in questo caso verrebbe segnalato un errore di doppia definizione delle procedure.

6.10 Menu Utility

Per eseguire un singolo comando del sistema operativo, anziché il comando **DOS Shell** del menu **File** (vedi paragrafo 6.5.5) è più opportuno utilizzare **Run DOS Command** del menu **Utility** (vedi figura 6.12).
Esso apre una finestra di dialogo in cui l'utente può inserire il comando MS-DOS voluto. Premendo *Enter*, questo comando viene eseguito e al termine si rientra automaticamente in ambiente QuickBASIC.
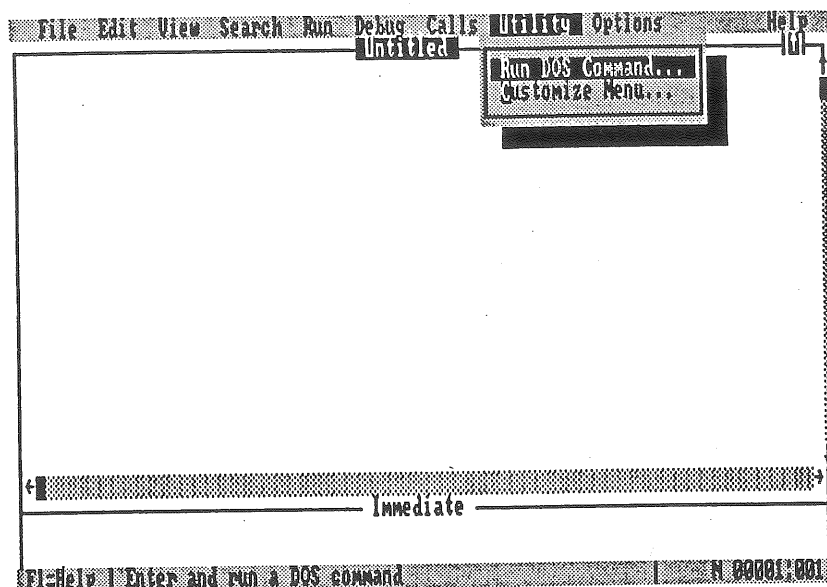
Figura 6.12

6.11 Controllo dell'esecuzione di un programma: menu Debug

Nel paragrafo 6.9 sono stati spiegati i comandi per lanciare l'esecuzione di un programma dall'inizio (**Start**), per riprenderla dopo una sospensione (**Continue**) o per predisporre una nuova esecuzione ripristinando le condizioni iniziali (**Restart**).

Quando il programma è pronto per l'esecuzione oppure è in attesa di essere ripreso dopo una sospensione, esso può essere seguito passo per passo a partire dal punto in cui si trova mediante opportuni comandi in forma abbreviata:

- il tasto *F8* esegue un'istruzione ogni volta che viene premuto; l'istruzione eseguita viene evidenziata sullo schermo;
- il tasto *F10* si comporta come *F8*, salvo che considera una chiamata di una procedura come un'unica istruzione evitando di esaminare passo a passo anche tutte le istruzioni del corpo della procedura.

Se in un determinato istante si vogliono esaminare i risultati fino a quel punto stampati, si usa il tasto *F4* (vedi paragrafo 6.8.1).

Il menu **Debug** mette a disposizione (vedi figura 6.13) una serie di comandi che aiutano a scoprire gli eventuali malfunzionamenti di un programma. A tale scopo è indispensabile interrompere l'esecuzione in determinati punti, seguire porzioni di programma passo per passo, esaminare via via i contenuti di certe variabili ed effettuare le opportune correzioni prima di riprendere l'esecuzione.
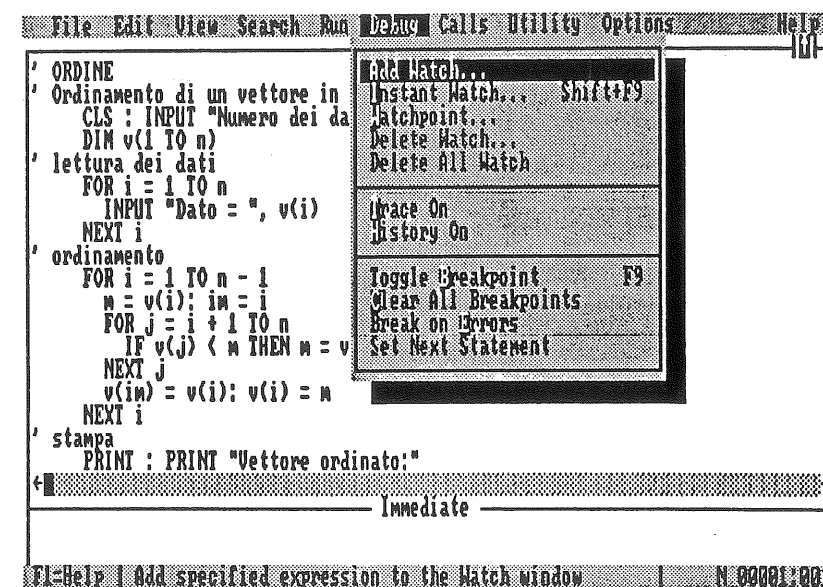
Figura 6.13

Conviene esaminare dettagliatamente l'andamento del programma solo nelle porzioni in cui si sospettano errori; a tale scopo si pongono dei punti di interruzione su linee opportune.

Il comando **Toggle Breakpoint**, con forma abbreviata *F9*, stabilisce un punto di sospensione sulla linea in cui si trova il cursore e la evidenzia in nero su sfondo bianco. L'utilizzo dello stesso comando su una linea già evidenziata rimuove la sospensione precedentemente inserita.

Una volta stabiliti dei "breakpoints", si può lanciare il programma con *Shift F5* o riavviarlo da un punto intermedio con *F5*: l'esecuzione proseguirà automaticamente fino al primo "breakpoint" e potrà da qui essere esaminata istruzione per istruzione mediante *F8* o *F10*.

L'esame dettagliato del programma può essere fatto solo se si possono controllare le variazioni di valore delle variabili sospette.
Il comando **Add Watch** apre una finestra di dialogo in cui l'utente può inserire il nome di una variabile o di una espressione, relativa alla procedura presente nella finestra di testo, di cui voglia esaminare il valore durante l'esecuzione del programma.
Le variabili e le espressioni scelte con **Add Watch** appaiono in una lista nella *finestra Watch* che viene aperta sopra alla finestra di testo: ciascuna di esse è preceduta dal nome della procedura in cui è definita e seguita dal valore attuale, che si vede cambiare durante l'esecuzione. Quando viene eseguita una procedura, tutte le variabili presenti nella finestra Watch che sono locali ad altre procedure non sono accessibili (`< Not Watchable >`).

Il comando **Instant Watch** (con forma abbreviata *Shift F9*) apre una finestra di dialogo; nella sottofinestra `Expression` viene presentato il nome della variabile su cui è posizionato il cursore nella finestra di testo oppure l'espressione evidenziata dall'utente e nella sottostante finestra `Value` compare il valore corrente di tale variabile o espressione.
Oltre a `< Cancel >` e `< Help >` è presente l'azione `< Add Watch >`, che è la scelta già predisposta e aggiunge nella finestra Watch la variabile o l'espressione indicata.
Questo comando costituisce un modo alternativo a **Add Watch** per inserire le variabili nella finestra Watch.

Il comando **Delete Watch** apre una finestra di dialogo contenente l'elenco delle variabili ed espressioni della finestra Watch; l'utente può posizionarsi su una di esse per eliminarla.

Il comando **Delete All Watch** elimina la finestra Watch con tutto il suo contenuto.

6.12 Altre modalità di chiamata di QBX

Oltre ai comandi

QBX

e

QBX *nome* /L *nomelib*

esistono altre modalità di attivazione dell'ambiente QuickBASIC.
Le più utili si possono riassumere nella seguente sintassi:

QBX [/RUN] *nome* [/L *nomelib*]

Se è presente solo *nome*, si entra in ambiente con il programma *nome* già caricato in memoria.

Se compare anche l'opzione **/RUN**, il programma viene mandato direttamente in esecuzione.

Se è presente l'opzione **/L**, oltre alla libreria standard viene utilizzata anche quella presente in *nomelib*.

Capitolo 7

Esempi di programmazione

In questo capitolo si presentano alcuni esempi di programmazione abbastanza semplici ma completi, al fine di mostrare l'uso pratico delle istruzioni Basic spiegate in precedenza e la loro organizzazione nella struttura di un programma.

7.1 Ordinamento di un vettore

Il programma ORDINE sistema in ordine di valore crescente gli elementi di un vettore di numeri in virgola mobile.

Inizialmente esso chiede all'utente il numero n dei dati ed i dati stessi, che vengono memorizzati nel vettore dinamico v. Quindi procede a riordinare gli elementi di v mediante un ciclo di n−1 passi, sistemando un elemento ad ogni passo. Al passo i-esimo determina mediante confronti successivi il minimo tra v(i), ... , v(n), ricordandone il valore in m e la posizione in im, e lo pone nella posizione i scambiandolo con v(i). L'elemento di valore massimo risulta automaticamente in v(n).

Listato del programma

```
' ORDINE
' Ordinamento di un vettore in ordine crescente
    CLS : INPUT "Numero dei dati = ", n
    DIM v(1 TO n)
```

```
' lettura dei dati
    FOR i = 1 TO n
      INPUT "Dato = ", v(i)
    NEXT i
' ordinamento
    FOR i = 1 TO n - 1
      m = v(i): im = i
      FOR j = i + 1 TO n
        IF v(j) < m THEN m = v(j): im = j
      NEXT j
      v(im) = v(i): v(i) = m
    NEXT i
' stampa
    PRINT : PRINT "Vettore ordinato:"
    FOR i = 1 TO n
      PRINT v(i);
    NEXT i: PRINT : END
```

7.2 Triangolo di Tartaglia

Il programma TARTAGLI calcola i coefficienti del triangolo di Tartaglia arrestandosi alla riga indicata dall'utente.

La prima riga è composta da due elementi entrambi uguali a 1.
Per $i > 1$ l'i-esima riga è composta da $i+1$ elementi. Il primo e l'ultimo sono costanti ed uguali a 1; ogni altro elemento si ottiene come somma di due elementi della riga $(i-1)$-esima: quello di ugual posizione e il precedente.

Il numero della riga finale viene letto e memorizzato nella variabile n. Il suo valore deve essere minore di 18 per non superare la capacità di rappresentazione degli interi in semplice precisione; per il calcolo con valori maggiori è necessario dichiarare m come vettore di tipo LNG e aumentarne la dimensione.

Gli elementi di ogni riga sono calcolati nel vettore m, sovrapponendoli di volta in volta a quelli della riga precedente.

L'elemento m(1) è posto a 1 una volta per tutte all'inizio del programma.
Per ogni riga i (da 1 a n), si pone a 1 anche m(i+1) e si esegue per j = i,i−1,. . .,2 l'istruzione m(j) = m(j) + m(j-1) che modifica il vettore m, sostituendo la nuova riga alla vecchia. (Si noti che per la prima riga questo ciclo su j non viene svolto.) Il calcolo si esegue all'indietro, partendo dalla penultima componente fino alla seconda, per non alterare anzitempo i valori che ancora devono essere utilizzati. Il vettore m così ottenuto viene poi stampato, prima di passare alla determinazione della riga successiva.

Listato del programma

```
' TARTAGLI
' Calcolo del triangolo di Tartaglia
    DIM m(1 TO 18): CLS
    INPUT "Numero righe da calcolare (<18) ", n
    PRINT : m(1) = 1
    FOR i = 1 TO n
      m(i + 1) = 1
      FOR j = i TO 2 STEP -1
        m(j) = m(j) + m(j - 1)
      NEXT j
      FOR j = 1 TO i + 1
        PRINT USING "#####"; m(j);
      NEXT j: PRINT
    NEXT i
    END
```

7.3 Radice quadrata

Il programma RADICE calcola la radice quadrata di un numero x fornito dall'utente, utilizzando il metodo iterativo

$$y_n = \frac{1}{2}(y_{n-1} + \frac{x}{y_{n-1}}) \quad \text{per } n = 1, 2, \ldots \quad \text{con } y_0 = 1$$

che approssima via via il valore cercato.

Il procedimento termina quando due approssimazioni y_n successive sono uguali.

Tutti i valori y_n sono calcolati nella medesima variabile y, modificando ad ogni iterazione il valore precedente, che viene salvato in yp per il confronto.

Listato del programma

```
' RADICE
' Calcolo radice di x con metodo iterativo
    INPUT "Valore di x = ", x
    y = 1: yp = -1
    DO WHILE y <> yp
      yp = y
      y = (y + x / y) / 2
      PRINT "valore approssimato "; y
    LOOP
    PRINT "Radice di "; x; " = "; y
    END
```

7.4 Valore di un polinomio in un punto

Il programma POLINOMI calcola il valore di un polinomio $c(x)$ di grado n in un punto x assegnato. Il calcolo è svolto mediante lo schema di Horner:

$$c(x) = (\cdots(c_n x + c_{n-1})x + \cdots + c_1)x + c_0$$

e può essere ripetuto per più polinomi e, per ogni polinomio, per più valori di x.

Inizialmente il programma chiede all'utente di introdurre il grado del polinomio, che viene assegnato alla variabile n, ed i suoi coefficienti, in ordine di grado crescente, che vengono memorizzati in un vettore dinamico c a n+1 componenti (numerate da 0 a n).

Alla linea 100 viene richiesto il punto in cui calcolare il polinomio; il dato fornito è considerato come variabile stringa x$.

Se l'utente introduce un dato effettivo, da x$ viene ricavato il valore numerico x mediante la funzione VAL. Un ciclo FOR...NEXT con passo -1 calcola in y il valore del polinomio secondo lo schema di Horner. Il programma ricicla poi alla linea 100, dove richiede un altro punto in cui valutare lo stesso polinomio.

Quando l'utente risponde alla richiesta di introdurre il punto battendo il solo tasto *Enter*, la stringa x$ risulta vuota. Il programma passa allora ad eseguire la linea successiva all'istruzione END IF.
Se l'utente desidera effettuare i calcoli con un altro polinomio, risponde poi alla relativa domanda battendo il tasto *1*; il vettore c viene cancellato per permetterne un dimensionamento diverso ed il programma viene rieseguito dall'inizio.
Se invece l'utente batte *Enter* o comunque un tasto diverso da *1*, il programma ha termine.

Listato del programma

```
' POLINOMI
' Calcolo del valore di un polinomio in un punto x
40    CLS : INPUT "Grado del polinomio = ", n
      DIM c(n)
      FOR i = 0 TO n
        PRINT "Coeff. di x^"; USING "##"; i;
        INPUT " = ", c(i)
      NEXT i
100   PRINT "P.to in cui calcolare il polinomio ";
      INPUT "(ENTER per finire) = ", x$
      IF x$ <> "" THEN
        x = VAL(x$): y = 0
' calcolo con lo schema di Horner
        FOR i = n TO 0 STEP -1
          y = y * x + c(i)
        NEXT i
        PRINT "Risultato = "; y
        GOTO 100
      END IF
```

```
    PRINT : PRINT "Altro polinomio [1] o ENTER ";
170 x$ = INKEY$: IF x$ = "" THEN 170
    IF x$ = "1" THEN ERASE c: GOTO 40
    END
```

7.5 Sostituzione di caratteri

Il programma ALFARF legge una o più stringhe p$ e per ognuna di esse determina una stringa corrispondente pf$, sostituendo ogni vocale di p$ con una terna di caratteri composta dalla vocale stessa, dalla lettera "f" e ancora dalla vocale. Il linguaggio delle stringhe pf$ è usato per gioco da molti bambini, che lo conoscono come "lingua farfallina" o "alfabeto farfallino".

Le vocali sono memorizzate nel vettore v$ mediante una istruzione READ che le preleva da una DATA.

Letta una parola p$, il programma ne estrae i caratteri ad uno ad uno in l$ servendosi della funzione MID$: se il carattere estratto è una consonante lo ricopia semplicemente in pf$, se è una vocale lo ricopia in pf$ aggiungendo la lettera "f" e la vocale stessa; al termine della scansione di p$ stampa la sua "traduzione" pf$.

L'utente può poi immettere un'altra parola p$ oppure battere solo *Enter* per porre fine al procedimento.

Listato del programma

```
' ALFARF
' Traduzione di parole in alfabeto farfallino
    DIM v$(1 TO 5)
    DATA a,e,i,o,u
    FOR i = 1 TO 5
      READ v$(i)
    NEXT i
100 INPUT "Parola (Enter per finire): ", p$
    IF p$ = "" THEN END
    pf$ = ""
```



```
    FOR k = 1 TO LEN(p$)
      l$ = MID$(p$, k, 1)
      FOR i = 1 TO 5
        IF l$ = v$(i) THEN
          pf$ = pf$ + l$ + "f"
          EXIT FOR
        END IF
      NEXT i
      pf$ = pf$ + l$
    NEXT k
    PRINT "Parola farfallina: "; pf$
    PRINT : GOTO 100
```

7.6 Grafico di una funzione

Il programma GRAFICO traccia il grafico della funzione

$$y = x \log |x|$$

in un intervallo $(-a, a)$, ove a è un valore fornito dall'utente.

A tale scopo calcola la funzione nei punti

$$x_i = -a + i\frac{a}{128} \quad \text{per } i = 0, 1, \ldots, 256$$

e congiunge i punti (x_i, y_i) ottenuti.

Il grafico può essere ripetuto per più valori di a.

La funzione è definita all'interno del programma mediante l'istruzione DEF FNy; cambiando opportunamemente tale definizione è possibile tracciare il grafico di una funzione diversa.

L'istruzione SCREEN 12 pone lo schermo in modo grafico per una scheda VGA.

I valori x_i e y_i sono calcolati sempre nelle stesse variabili x e y.

Si noti la presenza della frase ON ERROR all'inizio del programma, che rinvia alla linea 1000 in caso di errore. Se il programma si trova a

calcolare il valore y in un punto x in cui la funzione non è definita, viene stampata un'opportuna segnalazione e a y viene attribuito il valore 0 per la prosecuzione del tracciamento del grafico. La routine di trattamento dell'errore non prevede però il verificarsi di errori di altro tipo.

Listato del programma

```
' GRAFICO
' Grafico di x * log(abs(x))
      SCREEN 12
      ON ERROR GOTO 1000
' per un'altra funzione cambiare la frase seguente
      DEF FNy (x) = x * LOG(ABS(x))
30    CLS : INPUT "Intervallo da -a ad a : a = ", a
      WINDOW (-a, -a)-(a, a): CLS
      h = a / 128: x = -a: i = 0
40    y = FNy(x)
320   IF x > -a THEN LINE (xp, yp)-(x, y)
      xp = x: yp = y
      x = x + h
      i = i + 1: IF i < 257 THEN 40
      LOCATE 25, 1
      PRINT "Riesecuzione [1] o ENTER "
160   a$ = INKEY$: IF a$ = "" THEN 160
      IF a$ = "1" THEN 30 ELSE END
' routine d'errore
1000  LOCATE 1, 1: PRINT SPACE$(50); : LOCATE 1, 1
      PRINT "funzione non definita per x = "; x
      y = 0: RESUME 320
```

7.7 Grafico di una curva parametrica

Il programma TRACUR traccia il grafico di una spirale assegnata in modo parametrico:

$$\begin{cases} x = & e^{-0.1t}\cos t \\ y = & e^{-0.1t}\sin t \end{cases}$$

Il grafico viene rappresentato nel rettangolo

$$-a < x < a \qquad -b < y < b$$

ove i valori a e b sono forniti dall'utente, e può essere ripetuto in rettangoli diversi.

La tabulazione è eseguita in 1001 punti, per valori del parametro t tali che $0 \leq t \leq 30$, con passo 0.03.

Le definizioni parametriche della funzione vengono introdotte all'inizio del programma mediante le istruzioni `DEF FNx` e `DEF FNy`. Cambiando opportunamente le definizioni è possibile ottenere il grafico di una funzione con formule parametriche diverse.

Listato del programma

```
' TRACUR
' Grafico di una spirale assegnata in modo parametrico
      SCREEN 12
      DEF FNx (t) = COS(t) * EXP(-.1 * t)
      DEF FNy (t) = SIN(t) * EXP(-.1 * t)
40    CLS : PRINT "Tracciamento di una spirale ";
      PRINT "assegnata in modo parametrico"
      PRINT "nel rettangolo -a<x<a , -b<y<b ";
      PRINT "con 0<=t=<30"
      INPUT "a,b (valori consigl.: 1.5,1.3) "; a, b
      WINDOW (-a, -b)-(a, b): t = 0
      FOR i = 0 TO 1000
        x = FNx(t): y = FNy(t)
        IF i > 0 THEN LINE (xp, yp)-(x, y)
        t = t + .03: xp = x: yp = y
      NEXT i
      LOCATE 25, 1
      PRINT "Riesecuzione [1] o ENTER "
170   a$ = INKEY$: IF a$ = "" THEN 170
      IF a$ = "1" THEN 40
      END
```

7.8 Minimo comune multiplo

Il programma `MCM` calcola il minimo comune multiplo tra due interi positivi assegnati a e b mediante la formula

$$mcm(a,b) = \frac{a \cdot b}{MCD(a,b)}$$

Il massimo comun divisore viene calcolato con una delle due procedure `mcd1` e `mcd2`, a scelta dell'utente.

La `FUNCTION mcd1` calcola $MCD(a,b)$ con l'algoritmo euclideo così definito:

sia $a = q \cdot b + r \quad$ con $r < b$

$$\begin{array}{ll} MCD(a,b) = MCD(b,r) & \text{se } r \neq 0 \\ MCD(a,b) = b & \text{se } r = 0 \end{array}$$

La `FUNCTION mcd2` calcola $MCD(a,b)$ mediante l'algoritmo delle sottrazioni successive:

$$\begin{array}{ll} MCD(a,b) = MCD(a-b,b) & \text{se } a > b \\ MCD(a,b) = b & \text{se } a = b \end{array}$$

La scelta tra i due algoritmi viene effettuata rispondendo ad una domanda posta nel programma principale.

Si noti che entrambe le procedure operano su due variabili `a` e `b` in cui inizialmente vengono copiati i valori trasmessi dal programma principale. Infatti, poiché la trasmissione dei parametri è fatta per indirizzo, se le due procedure lavorassero direttamente sui parametri restituirebbero al programma principale dei valori modificati. (Le procedure potrebbero modificare i parametri se la trasmissione fosse fatta per valore.)

Si noti inoltre la presenza nel programma delle istruzioni `DECLARE`, che specificano le caratteristiche delle procedure.

Listato del programma

```
DECLARE FUNCTION mcd1% (aa%, bb%)
```



```
    DECLARE FUNCTION mcd2% (aa%, bb%)
' MCM
' Calcolo del mcm tra due numeri positivi con
' l'algoritmo euclideo o con sottrazioni successive
    DEFINT A-Z
    CLS : INPUT "Numeri: ", a, b
    PRINT "per l'algoritmo euclideo batti 1"
    PRINT "per l'algoritmo delle sottrazioni ";
    PRINT "successive batti 2"
    INPUT n
    IF n = 1 THEN
' con algoritmo euclideo
      m = mcd1(a, b)
    ELSE
' con algoritmo delle sottrazioni successive
      m = mcd2(a, b)
    END IF
    mcm = (a * b) / m
    PRINT "mcm = ("; a; ","; b; ") ="; mcm
    END

    FUNCTION mcd1 (aa, bb)
' Calcolo mcd tra due interi aa e bb
' con l'algoritmo euclideo
    a = aa: b = bb
    DO
      q = INT(a / b): r = a - b * q
      a = b: b = r
    LOOP UNTIL r = 0
    mcd1 = a
    END FUNCTION

    FUNCTION mcd2 (aa, bb)
' Calcolo mcd tra due interi aa e bb
' con l'algoritmo delle sottrazioni successive
    a = aa: b = bb
    WHILE a <> b
```

```
  IF a > b THEN a = a - b ELSE b = b - a
WEND
mcd2 = a
END FUNCTION
```

7.9 Fattorizzazione di un intero

Il programma FATTORI calcola i fattori primi di un intero positivo assegnato, con le relative molteplicità.

Detto n il numero da fattorizzare e posto i = 2, se n è divisibile per i il programma continua a incrementare la molteplicità m (inizialmente nulla) ed a sostituire a n il quoziente tra n e i.
Quando n non è più divisibile per i, il programma stampa i e la sua molteplicità m (naturalmente se i è stato trovato essere un fattore di n), quindi ripete le operazioni con un nuovo valore di i.

La prima volta i viene incrementato di una quantità in pari a 1, passando dal valore 2 a 3; i successivi incrementi in sono uguali a 2, per esaminare solo gli i di valore dispari.

Il valore di i continua ad essere incrementato fino a quando si mantiene minore della radice quadrata del valore corrente di n. Se, al termine, il valore di n risulta maggiore di 1, esso costituisce un ulteriore fattore primo del numero assegnato, con molteplicità 1. (Questo perché un numero può avere un solo fattore primo superiore alla sua radice quadrata.)

Listato del programma

```
' FATTORI
' Calcolo dei fattori primi di un numero n
' con le relative molteplicita'
    INPUT "n = ", n
    PRINT "valori", "molteplicita'"
    in = 1: i = 2
    DO WHILE i * i <= n
      m = 0
```



```
      DO WHILE n MOD i = 0
        m = m + 1: n = n \ i
      LOOP
      IF m > 0 THEN PRINT i, m
      i = i + in: in = 2
    LOOP
    IF n > 1 THEN PRINT n, 1
    END
```

7.10 Successione di Fibonacci

Il programma FIBONACC calcola gli elementi della successione di Fibonacci $\{fib_n\}$ definiti da

$$fib_n = fib_{n-1} + fib_{n-2}$$

con $fib_0 = 1$ e $fib_1 = 1$.
Il calcolo è svolto per $n = 2, \ldots, 19$ utilizzando prima un metodo ricorsivo e poi un metodo iterativo e fornendo, per confronto, i relativi tempi di elaborazione.

Dapprima il programma usa la funzione ricorsiva fib, definita nello stesso modo della successione.
Successivamente svolge il calcolo con il metodo iterativo definito dalle seguenti formule:
per $n = 2, \ldots$

$$fib_n = i_{n-1} + j_{n-1}$$
$$i_n = fib_n$$
$$j_n = i_{n-1}$$

con

$$i_1 = 1 \quad \text{e} \quad j_1 = 1.$$

Per implementare questo metodo bastano tre variabili i, j e k: in k si determina l'elemento fib_n come somma dei due elementi precedenti contenuti in i e j.

Per valutare il tempo di calcolo di ciascun metodo viene usata la funzione TIMER, che restituisce il numero dei secondi trascorsi dalla mezzanotte. Dal valore di TIMER al termine dei calcoli viene sottratto

il valore di TIMER memorizzato all'inizio dei calcoli: si ottiene così il tempo speso per ogni algoritmo.
Il confronto dei due tempi mostra che il metodo iterativo è più rapido del metodo ricorsivo. Questo è vero in generale e perciò i metodi ricorsivi vanno utilizzati solo quando non esista una definizione più semplice per la funzione di cui occorre calcolare il valore.

Listato del programma

```
    DECLARE FUNCTION fib& (n&)
' FIBONACCI
' Calcolo numeri di Fibonacci con due metodi
    DEFLNG A-X
    PRINT "Calcolo Fibonacci: metodo ricorsivo"
    z = TIMER
    FOR n = 2 TO 19
      PRINT "   fib("; USING "##"; n;
      PRINT ") ="; USING "####"; fib(n);
    NEXT n: PRINT
    PRINT "tempo di calcolo "; TIMER - z
    PRINT "Calcolo Fibonacci: metodo iterativo"
    i = 1: j = 1: z = TIMER
    FOR n = 2 TO 19
      k = i + j: j = i: i = k
      PRINT "   fib("; USING "##"; n;
      PRINT ") ="; USING "####"; k;
    NEXT n: PRINT
    PRINT "tempo di calcolo "; TIMER - z
    END

    FUNCTION fib (n)
    IF n <= 1 THEN fib = 1: EXIT FUNCTION
    fib = fib(n - 1) + fib(n - 2)
    END FUNCTION
```


7.11 Inserimento e ricerca dicotomici

Il programma RICERCA permette di inserire e ricercare dati in una tabella ordinata. Esso lavora su due vettori: il vettore t contenente le chiavi di ricerca ed il vettore x contenente, nelle posizioni corrispondenti, i dati associati a tali chiavi; in altre parole il dato associato alla chiave t(lp) è x(lp). Inizialmente i due vettori sono vuoti.

Il programma prevede tre tipi di richieste da parte dell'utente:

1) fornendo i dati (1,c,d) l'utente chiede di inserire la chiave c nella posizione di t che le compete secondo l'ordinamento crescente e di porre il dato d nella posizione corrispondente di x;

2) fornendo i dati (2,c,0) l'utente chiede di ricercare la posizione di t in cui è contenuta la chiave c e di restituire il dato associato;

3) fornendo i dati (3,0,0) l'utente termina l'esecuzione del programma.

Tanto l'inserimento che la ricerca nella tabella t sono realizzati con metodo dicotomico.

Si confronta la chiave c con quella situata nel punto medio lp della tabella t.
Se t(lp)=c, nel caso 1) il programma segnala che la chiave è già presente e le assegna il nuovo dato dopo aver visualizzato il precedente; nel caso 2) fornisce come risultato il dato x(lp).
Se t(lp) $\neq$ c si stabilisce in quale delle due metà della tabella va cercata od inserita c, si modificano opportunamente gli indici li e ls che delimitano la porzione di tabella da prendere in esame e si torna a confrontare c con il nuovo elemento centrale.

L'algoritmo termina quando li $>$ ls.
Nel caso 1) a questo punto li indica la posizione in cui la chiave c va inserita nel vettore t; è perciò necessario traslare in t tutti gli elementi dalla posizione li in avanti per far posto alla nuova chiave e analoga operazione va ovviamente compiuta sul vettore x. Il numero n delle chiavi memorizzate viene aumentato di una unità.

Nel caso 2) si conclude invece che la chiave non è presente nella tabella.

Listato del programma

```
' RICERCA
' Ricerca e inserimento dicotomici
    DEFINT A-Z: DIM t(100), x(100)
    CLS : n = 0
    PRINT "Comando inser.  1,c,d  c=chiave d=dato"
    PRINT "Comando ricerca 2,c,0"
    PRINT "Comando fine    3,0,0"
70  INPUT "Comando ", i, c, d
    ls = n: li = 1
    ON i GOTO 100, 100, 200
    PRINT "errore": GOTO 70
' algoritmo dicotomico
100 DO
      lp = (ls + li) \ 2
      IF c = t(lp) THEN
        IF i = 1 THEN
          PRINT "chiave gia' assegnata a "; x(lp)
          x(lp) = d: GOTO 70
        ELSE
          PRINT "dato = "; x(lp)
        END IF
        GOTO 70
      END IF
      IF c > t(lp) THEN
        ls = lp - 1
      ELSE
        li = lp + 1
      END IF
    LOOP UNTIL li > ls
    IF i = 1 THEN
      FOR j = n TO li STEP -1
        t(j + 1) = t(j): x(j + 1) = x(j)
      NEXT j
```



```
      t(li) = c: x(li) = d: n = n + 1
    ELSE
      PRINT "chiave inesistente"
    END IF
    GOTO 70
' fine
200 END
```

7.12 Risoluzione di equazioni diofantee

Il programma DIOFANTE ricerca le soluzioni intere x e y dell'equazione lineare

$$ax + by = 1 \tag{7.1}$$

ove a e b sono due numeri interi positivi primi tra loro forniti dall'utente.

È noto che, dati a e b interi, il problema

$$ax + by = k$$

ammette come soluzione una coppia (x, y) di valori interi se e solo se k è il massimo comun divisore tra a e b.
Nella situazione presentata, poiché a e b sono primi tra loro, l'equazione (7.1) ammette sicuramente una soluzione.
Si può facilmente verificare poi che ogni coppia del tipo

$$(x + nb, y - na)$$

con n intero è pure soluzione dell'equazione.

L'algoritmo utilizzato è un'estensione dell'algoritmo euclideo per il calcolo del massimo comun divisore.

I due valori forniti dall'utente vengono assegnati alle variabili `a` e `b` e copiati anche in `a1` e `b1`.

Si determinano poi quattro coefficienti `x`, `y`, `z` e `u` medianti i quali si possano esprimere `a1` e `b1` come combinazione lineare di `a` e `b`, cioè tali che tra i valori delle variabili sussistano le eguaglianze

$$\text{a1} = \text{a} * \text{z} + \text{b} * \text{u} \qquad (7.2)$$

$$\text{b1} = \text{a} * \text{x} + \text{b} * \text{y} \qquad (7.3)$$

Inizialmente si pongono perciò x e u uguali a 0, y e z uguali a 1.

Successivamente si applica l'algoritmo euclideo per il calcolo del MCD tra a1 e b1 e ad ogni iterazione si cambiano i valori di x, y, z e u in modo tale che al variare di a1 e b1 le relazioni (7.2) e (7.3) continuino a rimanere valide.
Si indichino con $\text{a1}^{(i)}$, $\text{b1}^{(i)}$, $\text{z}^{(i)}$, $\text{u}^{(i)}$, $\text{x}^{(i)}$ e $\text{y}^{(i)}$ i valori delle variabili al passo i-esimo dell'algoritmo euclideo. Poiché ad ogni passo a1 e b1 vengono modificati in modo che:

$$\begin{aligned}
\text{a1}^{(i)} &= \text{b1}^{(i-1)} \\
\text{b1}^{(i)} &= \text{r} = \\
&= \text{a1}^{(i-1)} - \text{q} * \text{b1}^{(i-1)} = \\
&= \text{a} * (\text{z}^{(i-1)} - \text{q} * \text{x}^{(i-1)}) + \text{b} * (\text{u}^{(i-1)} - \text{q} * \text{y}^{(i-1)})
\end{aligned}$$

i valori di z, u, x e y dovranno a loro volta così essere modificati per mantenere l'invarianza delle (7.2) e (7.3):

$$\begin{aligned}
\text{z}^{(i)} &= \text{x}^{(i-1)} \\
\text{u}^{(i)} &= \text{y}^{(i-1)} \\
\text{x}^{(i)} &= \text{z}^{(i-1)} - \text{q} * \text{x}^{(i-1)} \\
\text{y}^{(i)} &= \text{u}^{(i-1)} - \text{q} * \text{y}^{(i-1)}
\end{aligned}$$

L'algoritmo si arresta quando b1=1=$MCD(a,b)$. A questo punto i valori x e y sono una delle soluzioni cercate.

Se i dati forniti dall'utente non sono primi tra loro, nel corso del calcolo del massimo comun divisore si troverà un resto nullo con b1$\neq$ 1. In tal caso il programma emette una segnalazione e poi chiede nuovi dati.

Listato del programma

```
' DIOFANTE
' Ricerca soluzioni intere di un'equazione lineare
' Se la coppia x,y e' una soluzione, la coppia
' x+n*b,y-n*a con n intero e' una soluzione
   DEFINT A-Z: CLS
   PRINT "Ricerca dei valori x e y interi ";
   PRINT "tali che a*x+b*y=1"
50 PRINT "a,b interi positivi primi fra loro ";
   INPUT ": ", a, b
   a1 = a: b1 = b
   x = 0: y = 1: z = 1: u = 0: PRINT
   PRINT "   a1    b1     q     x     y     z     u"
70 q = a1 \ b1: r = a1 - q * b1
   PRINT USING "#####"; a1; b1; q; x; y; z; u
   IF r = 0 THEN
     PRINT "a e b non primi fra loro, MCD = "; b1
     GOTO 50
   END IF
   a1 = b1: b1 = r
   t = x: x = z - q * x: z = t
   t = y: y = u - q * y: u = t
   IF b1 > 1 THEN 70
   PRINT : PRINT "x = "; x; : PRINT "  y = "; y
   PRINT "verifica: a*x+b*y = "; a * x + b * y
   END
```

7.13 Divisione tra polinomi in Zp

Il programma DIVISION calcola il quoziente e il resto di una divisione tra polinomi a coefficienti in Z_p, classe di resti modulo p, ove p è un numero primo fornito dall'utente e memorizzato in una variabile con lo stesso nome.

Oltre a p, il programma richiede i gradi n e m dei polinomi dividendo e divisore, ed i relativi coefficienti che memorizza rispettivamente nei

vettori s e q in ordine di grado crescente.

Nel vettore in vengono poi calcolati tutti i possibili quozienti delle divisioni f/q(m), per f = 1, ... , p−1. Infatti si pone in(f) = i se f è il prodotto di i per q(m).
Si noti che, essendo p primo, tutti i quozienti sono distinti e assumono anch'essi tutti i valori tra 1 e p−1.

Il calcolo dei polinomi quoziente e resto è effettuato con l'usuale algoritmo della divisione tra polinomi algebrici, utilizzando i valori preventivamente calcolati nel vettore in.
I coefficienti del polinomio quoziente, di grado n−m, vengono calcolati nel vettore r a partire dal coefficiente di grado maggiore. I coefficienti del polinomio resto, di grado m−1, sono calcolati nel vettore s, che viene dunque modificato nel corso delle operazioni.

Al termine, vengono stampati i coefficienti dei due polinomi risultato, in ordine di grado crescente.

Listato del programma

```
' DIVISIONE
' Calcola quoziente e resto di polinomi
' a coefficienti in Zp
    DEFINT A-Z
    DIM s(20), q(20), r(20), in(20): CLS
    INPUT "Classe di resti mod p (primo): p = ", p
60  INPUT "Grado dei polinomi s e q ", n, m: PRINT
    FOR i = 0 TO n
      PRINT "coeff. in s di x^"; USING "#"; i;
      INPUT " ", s(i)
    NEXT: PRINT
    FOR i = 0 TO m
      PRINT "coeff. in q di x^"; USING "#"; i;
      INPUT " ", q(i)
    NEXT: PRINT
    IF q(m) = 0 THEN PRINT "Errore": GOTO 60
' calcolo dei quozienti f/q(m) con f = 1,...,p-1
    FOR i = 1 TO p - 1
      f = (i * q(m)) MOD p: in(f) = i
    NEXT i
' divisione
    nr = n - m: l = n
    FOR i = nr TO 0 STEP -1
      r(i) = in(s(l))
      FOR j = 0 TO m
        s(j + i) = (s(j + i) - r(i) * q(j)) MOD p
        IF s(j + i) < 0 THEN
          s(j + i) = s(j + i) + p
        END IF
      NEXT j: l = l - 1
    NEXT i
    PRINT "quoziente ";
    FOR i = 0 TO nr
      PRINT USING "###"; r(i);
    NEXT i: PRINT
    PRINT "resto     ";
    FOR i = 0 TO m - 1
      PRINT USING "###"; s(i);
    NEXT i: PRINT
    END
```


7.14 Sviluppo in serie di 1/(1-p(x))

Il programma SVILUPPO calcola i coefficienti dello sviluppo in serie di $\frac{1}{1-p(x)}$, ove $p(x)$ è un polinomio con termine noto diverso da 1. Lo sviluppo è arrestato ad un ordine k assegnato.

Sia $p(x) = p_0 + p_1x + p_2x^2 + \cdots + p_nx^n$ il polinomio dato.
Posto $t = p_0$, si ha:

$$\frac{1}{1-p(x)} = \frac{1}{t(1 - \frac{p_1}{t}x - \frac{p_2}{t}x^2 - \dots - \frac{p_n}{t}x^n)}$$

Si consideri lo sviluppo in serie di $\frac{t}{1-p(x)}$:

$$
\begin{aligned}
\frac{t}{1-p(x)} &= \frac{1}{1-p'_1x-p'_2x^2-\cdots-p'_nx^n} = \\
&= r_0+r_1x+r_2x^2+\cdots+r_kx^k+\cdots
\end{aligned}
\qquad (7.4)
$$

ove $p'_i = \frac{p_i}{t}$ per $i = 1,\ldots,n$.

Per il principio di identità, se $k > n$ vale la relazione

$$
\begin{aligned}
1 &= r_0 + x(-r_0p'_1+r_1) + x^2(-r_0p'_2-r_1p'_1+r_2)+\cdots+ \\
&+x^n(-r_0p'_n-r_1p'_{n-1}-\cdots-r_{n-1}p'_1+r_n)+\cdots+ \\
&+x^k(-r_{k-n}p'_n-\cdots-r_{k-1}p'_1+r_k)+\cdots
\end{aligned}
$$

altrimenti

$$
\begin{aligned}
1 &= r_0 + x(-r_0p'_1+r_1) + x^2(-r_0p'_2-r_1p'_1+r_2)+\cdots+ \\
&+x^k(-r_0p'_k-r_1p'_{k-1}-\cdots-r_{k-1}p'_1+r_k)+\cdots
\end{aligned}
$$

Il coefficiente i-esimo dello sviluppo (7.4), per $i = 1,\ldots,k$, è dato pertanto dalla formula

$$
r_i = \sum_{j=0}^{i-1} r_jp_{i-j} \quad \text{con } i-j \leq n. \qquad (7.5)
$$

I coefficienti dello sviluppo di $\frac{1}{1-p(x)}$ si ottengono dividendo per t i valori r_i così determinati.

Inizialmente il programma richiede all'utente di fornire il grado del polinomio $p(x)$, che viene posto nella variabile n, i suoi coefficienti, che vengono memorizzati nel vettore p in ordine di grado crescente, e il numero dei termini dello sviluppo in serie, assegnato alla variabile nt. (Pertanto lo sviluppo sarà arrestato all'ordine $k = \mathtt{nt} - 1$.)

Dopo aver posto t=1-p(0) e diviso per t gli elementi di p, il programma procede poi a calcolare i coefficienti dello sviluppo (7.4) nel vettore r, in ordine di grado crescente.
Posto r(0) uguale a 1, invece di determinare le altre componenti una alla volta mediante la (7.5), si applica un algoritmo di calcolo che ad ogni passo agisce su n componenti consecutive sommando un addendo della (7.5) a ciascuna di esse. Più precisamente, osservato che all'inizio tutte le componenti di r sono nulle esclusa r(0), al passo i-esimo dell'algoritmo (i = 1, ..., nt−1) si sommano ordinatamente ai valori già presenti nelle componenti r(i), ..., r(i+n-1) i contributi ottenuti moltiplicando r(i-1) per tutti i coefficienti di p. Si noti che in tal modo la componente r(i) risulta completamente determinata al passo i-esimo e può essere pertanto utilizzata al passo successivo.

Infine si dividono le componenti di r per t e si stampano i risultati, che costituiscono i coefficienti dello sviluppo richiesto.

Listato del programma

```
' SVILUPPO
' Sviluppo in serie di 1/(1-p(x))
     DIM p(20), r(100)
40   CLS : INPUT "grado del polinomio = ", n
     FOR i = 0 TO n
       IF i = 0 THEN
         INPUT "termine noto (diverso da 1)  = ", p(0)
       ELSE
         PRINT "coeff. del termine x^"; USING "#"; i;
         INPUT " = ", p(i)
       END IF
     NEXT i
     t = 1 - p(0)
     IF t = 0 THEN PRINT "sviluppo impossibile": GOTO 40
     INPUT "numero termini sviluppo in serie = ", nt
     FOR i = 1 TO n
       p(i) = p(i) / t
     NEXT i
```

```
r(0) = 1
FOR i = 1 TO nt - 1
  l = i
  FOR j = 1 TO n
    r(l) = r(l) + p(j) * r(i - 1): l = l + 1
  NEXT j
NEXT i
FOR i = 0 TO nt - 1
  r(i) = r(i) / t: PRINT USING "#####.###"; r(i);
NEXT i: PRINT
END
```


Appendice A

Il software ESERCIZI

Il software ESERCIZI è stato appositamente predisposto come supporto didattico per il corso di "Laboratorio di Programmazione" che si tiene presso il corso di laurea in Matematica dell'Università degli Studi di Milano.
Esso funziona su PC dotati dell'ambiente QuickBASIC e di scheda grafica VGA, CGA o Hercules ed è composto da un insieme di programmi di diversa natura, costituenti però un'unica struttura. Alcuni sono algoritmi che aiutano lo studente a familiarizzarsi con il linguaggio Basic, con le tecniche fondamentali della programmazione e con i problemi che si presentano nel calcolo in virgola mobile; altri programmi riguardano specifici argomenti di algebra, analisi, fisica e possono quindi essere utilizzati anche come strumento per lo studio di queste materie.

A.1 Uso del software

Per utilizzare ESERCIZI basta immettere a livello MS-DOS il comando

ESERCIZI

Viene visualizzato un menu principale (v. fig. A.1), che presenta un elenco di sottomenu corrispondenti ai sottoinsiemi in cui sono raggruppati i programmi.

L'utente sceglie quello desiderato, scrivendone il nome seguito da *Enter*;

Figura A.1

il sistema passa allora a presentare l'elenco dei programmi disponibili nel sottomenu. (Le figure A.2, A.3, A.4, A.5 illustrano i quattro sottomenu attuali.)

Con la stessa tecnica l'utente effettua la scelta tra i programmi: quello indicato viene allora eseguito.

Nella maggioranza dei casi il programma richiede all'utente di introdurre dei dati: i dati numerici devono essere immessi secondo le regole previste dall'istruzione INPUT, le espressioni che definiscono funzioni devono essere espressioni valide del Basic.

Al termine dei calcoli con i dati forniti, il programma solitamente emette sull'ultima riga dello schermo un messaggio che, a seconda dei casi, propone diverse possibilità di continuazione: riesecuzione con altri dati, cambio di funzione, ritorno al sottomenu (v. fig. A.6). L'utente deve rispondere battendo il tasto corrispondente alla continuazione desiderata. Poiché questa lettura è effettuata dal programma mediante la funzione INKEY$, deve essere premuto un unico tasto,

altrimenti quelli battuti in eccedenza sarebbero presi come risposta alle letture successive.

A qualunque livello di scelta (nel programma, nel sottomenu o nel menu principale), premendo solo *Enter* si ritorna al livello precedente. Dal menu principale si rientra al sistema operativo in modo automatico se durante la sessione di lavoro con ESERCIZI non si sono verificate interruzioni nell'esecuzione dei programmi; in caso contrario compare il messaggio `Press any key to continue` e, dopo aver premuto un tasto qualunque, ci si ritrova in ambiente QuickBASIC: è necessario allora utilizzare il comando Exit del menu File per tornare a livello MS-DOS.

Figura A.2

```
******************************************************************************
*****                                                                   *****
*****                      ESERCIZI DI ALGEBRA                          *****
*****                                                                   *****
******************************************************************************

Esercizi di algebra disponibili:

applica        studio dell'applicazione r*a in Zn
associativo    associativita' dell'operatore x+k*y in Zn
automorfismi   gruppo degli automorfismi di un gruppo ciclico
cicli          scomposizione di sostituzioni in cicli disgiunti
diretto        prodotto diretto di due gruppi ciclici
eqgr           equazione in un gruppo ciclico
generatori     generatori di un gruppo ciclico
modn           costruzione delle tavole delle operazioni mod n
operatore      analisi delle proprieta' di un operatore in Zn

Scrivi il nome dell'esercizio scelto oppure batti ENTER █
```

Figura A.3

```
******************************************************************************
*****                                                                   *****
*****                      ESERCIZI ELEMENTARI                          *****
*****                                                                   *****
******************************************************************************

Esercizi disponibili:

appder      approssimazione della derivata di cos(x)
divisione   quoziente e resto di polinomi in Zp
err         calcolo della precisione di macchina e della base
expx        sviluppo in serie della funzione esponenziale
fattori     calcolo dei fattori primi di un numero
fibonacci   calcolo dei numeri di Fibonacci
grafico     grafico di x*sin(1/x)
mcd         massimo comun divisore
polinomio   tabulazione di un polinomio
radice      calcolo della radice quadrata
sviluppo    sviluppo in serie di 1/(1-p), p=polinomio
tartaglia   triangolo di Tartaglia
tracur      grafico di una spirale assegnata in modo parametrico

scrivi l'esercizio scelto o ENTER per visualizzare i corsi disponibili █
```

Figura A.4


```
******************************************************************************
*****                                                                   *****
*****                    ALGORITMI INTERESSANTI                         *****
*****                                                                   *****
******************************************************************************

campo       linee di forza di un campo elettrostatico in un piano
diofante    soluzioni intere di una equazione lineare
fft         trasformata discrete di Fourier
ordine      ordinamento di n numeri
perm        generazione delle permutazioni
primi       calcolo di numeri primi
ricerca     ricerca tabellare dicotomica

Scrivi il nome dell'esercizio scelto oppure batti ENTER █
```

Figura A.5
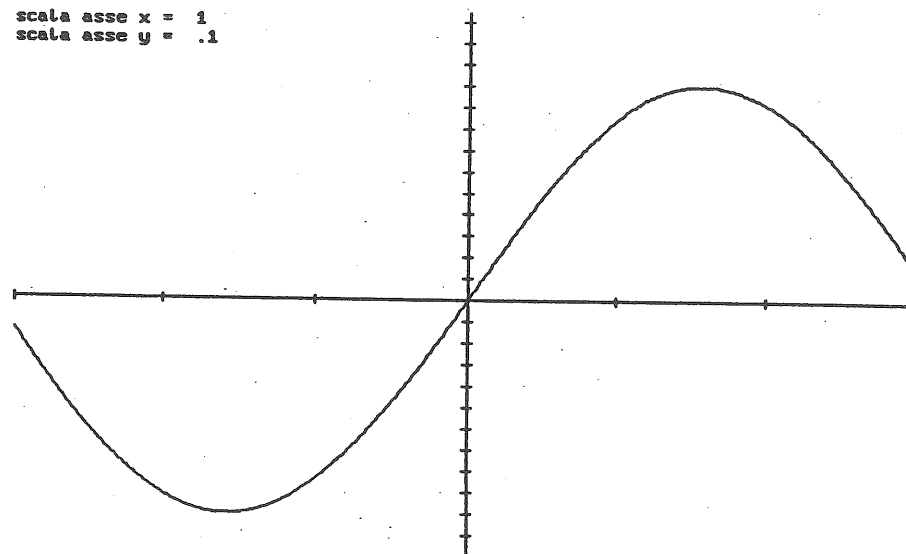
Figura A.6

A.2 Problemi d'uso

Se l'utilizzo di ESERCIZI è fatto rispondendo sempre correttamente alle domande presentate e introducendo i dati senza errori, il passaggio da un programma all'altro avviene senza inconvenienti e al termine si ritorna automaticamente in ambiente MS-DOS.

I problemi che si possono presentare sono dovuti essenzialmente all'introduzione di dati errati o all'interruzione volontaria di un programma mediante *Ctrl Break* per esame o modifica del sorgente. Qui di seguito si espongono le operazioni necessarie per far fronte alle situazioni anomale più frequenti.

1) Parecchi programmi richiedono all'utente di introdurre l'espressione della funzione su cui essi devono operare.

   a) Se l'espressione non è sintatticamente corretta, viene segnalato un errore in una finestra di dialogo mentre nella finestra di testo il cursore è posizionato sulla funzione errata. Si procede allora come segue:

      - si preme *Enter* per cancellare la segnalazione;
      - si corregge l'espressione della funzione puntata dal cursore;
      - con *Shift F5* si rilancia normalmente l'esecuzione.

      Quando poi si esce dal programma per passare ad un altro, QuickBASIC chiede se quello attualmente in memoria deve essere salvato: per proseguire basta rispondere con *Enter*.

   b) Se si interrompe il programma con *Ctrl Break*, basta premere *F5* per riprenderlo dal punto interrotto o *Shift F5* per rieseguirlo dall'inizio. Si faccia attenzione che un *Ctrl Break* dato quando il programma presenta le possibili alternative di continuazione viene preso in considerazione solo dopo che l'utente ha battuto il carattere di risposta.

2) Altri programmi utilizzano istruzioni grafiche. Essi possono essere ripresi correttamente dopo una interruzione solo se è stata

introdotta in testa al programma stesso l'istruzione SCREEN relativa alla scheda grafica in uso.

   Il programma va obbligatoriamente rieseguito dall'inizio con *Shift F5*; quando poi si passa ad un altro, alla richiesta se salvare il testo modificato è necessario rispondere negativamente, poiché ESERCIZI è protetto da scrittura. Se si vuole conservare copia del file modificato, si veda il punto 3).

3) I programmi che non richiedono all'utente l'espressione di una funzione possono essere rielaborati per ottenerne versioni modificate.

   Dopo aver richiamato ESERCIZI ed aver scelto il programma da modificare, lo si interrompe con *Ctrl Break* e si apportano le variazioni desiderate. Se il programma utilizza la grafica, è necessario inserire l'istruzione SCREEN opportuna.

   A questo punto è indispensabile salvare la nuova versione con un nome diverso da quello originale ma nello stesso direttorio.

   Il programma viene eseguito con *Shift F5*. Quando esso presenta le diverse alternative di continuazione, rispondendo *Enter* viene ripresentato il sottomenu da cui è stato scelto il programma originale. La versione modificata può essere scelta dallo stesso sottomenu anche se il suo nome non compare nell'elenco.

4) Gli stessi programmi del punto 3) possono essere prelevati dalla struttura di ESERCIZI per funzionare in modo indipendente. In tal caso l'utente deve eliminare dal testo tutte le istruzioni che in qualche modo legano il programma scelto con la struttura (per esempio le CHAIN e le RUN).

   Si tenga inoltre presente che ESERCIZI si avvale di alcune procedure inserite in una libreria che viene richiamata automaticamente.

   Se il programma isolato utilizza una di queste procedure, dopo averlo caricato in QuickBASIC è necessario caricare, con il comando Load, anche il modulo della libreria, il cui pathname è C:\MATR\SERVIZI\SERVIZI.BAS.

I programmi che richiedono all'utente l'espressione di una funzione non sono invece facilmente modificabili, poiché composti da due moduli. Un modulo è la parte operativa del programma; l'altro richiama una procedura (`genfu`), che prima genera un nuovo programma in un file temporaneo, ricopiandovi la parte operativa e accodandole una FUNCTION che calcola la funzione, e poi lancia l'esecuzione del programma generato.

Gli utenti interessati a modificare questi programmi devono pertanto esaminare attentamente la procedura `genfu`, che fa parte del modulo C:\MATR\SERVIZI\SERVIZI.BAS.